



**CRAY-1® AND CRAY X-MP
COMPUTER SYSTEMS**

**FORTRAN (CFT)
REFERENCE MANUAL**

SR-0009

Copyright© 1976, 1977, 1978, 1979, 1980, 1981, 1982, 1983,
1984 by CRAY RESEARCH, INC. This manual or parts thereof
may not be reproduced in any form without permission of
CRAY RESEARCH, INC.

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

<u>Revision</u>	<u>Description</u>
	July, 1976 - Preliminary distribution, Xerox copies
A	February, 1977 - First printing. Since changes are very extensive, they are not noted by change bars.
B	November, 1977 - Second printing. Since this represents a complete rewrite, changes are not noted by change bars.
C	April, 1978 - Updates the manual to be in full agreement with the April, 1978 release of the CRAY-1 FORTRAN Compiler (CFT) Version 1.01.
C-01	July, 1978 - Included in this change packet, which brings the manual into agreement with the FORTRAN Compiler Version 1.02, is a new description of listable output, changes to the type statements, the addition of several utility procedures, and several new messages.
C-02	October, 1978 - This change packet brings the manual into agreement with the FORTRAN compiler, Version 1.03. It includes the FLOWTRACE directive, new CFT messages, DO-loop table list option, and the ABORT subroutine.
D	January, 1979 - Reprint. This printing brings the manual into agreement with the FORTRAN compiler, Version 1.04. Major changes include DO-variable usage; addition of ERR and END to the control information list; scheduler directives; the TRUNC parameter on the CFT card; M, R, and W compiler options; vector and code generation information with intrinsic functions and utility procedures; new subroutines ERREXIT, REMARK2, and TRBK; and new CFT messages.
D-01	April, 1979 - This change packet brings the manual into agreement with version 1.05 of the FORTRAN compiler. Major changes include the alternate return feature, upper and lower bounds of DIMENSION declarators, and the NAMELIST statement.

- E April, 1979 - This revision is the same as Revision D with change packet D-01 incorporated.
- E-01 July, 1979 - This change packet brings the manual into agreement with version 1.06 of the CFT compiler. Major changes include conditional block statements ELSE IF, block IF, ELSE, and END IF; a new CFT directive BOUNDS; a new compiler option, O; a debugging utility, SYMDEBUG; and new CFT messages. Minor changes include clarification of Boolean arithmetic concepts and the introduction of dynamic memory allocation.
- E-02 July, 1979 - This change packet corrects a technical error appearing on page 6-6 of the E-01 change packet.
- It also replaces pages inadvertently deleted by the E-01 change packet.
- E-03 December, 1979 - This change packet brings the manual into agreement with version 1.07 of the CFT compiler. Major changes include a symbolic debug package, enabled by the CFT control statement option Z; utility procedures that permit or prohibit floating-point interrupts and that determine the current floating-point interrupt mode; an enhancement to the editing process that allows D, E, F, G, and O format specifications to edit both real and double-precision list items; and reprieve processing routines.
- F December, 1979 - This reprint includes change packets E-01, E-02, and E-03. It contains no other changes.
- F-01 April, 1980 - This change packet brings the manual into agreement with version 1.08 of the CFT compiler. Major changes include lower-case letters in the CFT character set, character constants, the POINTER statement, full implementation of reprieve processing, new subprograms REMARKF and DUMPJOB, new CFT messages, and unblocked I/O.
- G May, 1980 - This revision is the same as Revision F with change packet F-01 incorporated.
- G-01 October, 1980 - This change packet brings the manual into agreement with version 1.09 of the CFT compiler. Major changes include arithmetic constant expressions; the IMPLICIT NONE statement; the implementation of the PAUSE statement; sequential, direct, and random access; file identifiers in input/output statements; the INQUIRE, OPEN, and CLOSE statements; further clarification on vector operations; page header lines on listable output; a new flowtrace routine, FLODUMP; a new CFT parameter, AIDS; and new CFT messages.

- H August, 1981 - Rewrite. With this printing, the manual has been completely reorganized and updated to agree with version 1.10 of the CFT compiler. Major changes include adherence to ANSI X3.9-1978 (FORTRAN 77), including the character data type and the generic function feature and adding list-directed I/O. Other miscellaneous changes were also added. Changes are not noted by change bars. All previous versions are obsolete.
- H-01 August, 1982 - This change packet brings the manual into agreement with version 1.10 of the CFT compiler. Major changes include adding to the comment lines description; new intrinsic function names; new internal file restrictions, changing the INQUIRE table, the OPEN table, and the CLOSE table; adding to the CLOSE statement description and the NAMELIST statement description; moving time functions, Boolean functions, and vectorization aids from Appendix C to Appendix B; new CFT messages; and the Hollerith format specification.
- I November, 1982 - This revision is the same as Revision H with change packet H-01 incorporated.
- J April, 1983 - This reprint with revision brings the manual into agreement with version 1.11 of the CFT compiler. The formats of the following have changed: character substring, CHARACTER type statement, COMMON statement, FORMAT statement, CALL statement, SUBROUTINE statement, RETURN statement, and INT24 directive. The following are additions: DATA statement restrictions; information to program control statements and input/output statements; user control subroutine; the MAXBLOCK and INT parameters on the CFT control statement; optimization options; the INT64 integer control directive; the multiply/divide directives (FASTMD, SLOWMD); the optimization directives NO SIDE EFFECTS, ALIGN, NOIFCON, and RESUMEIFCON; and vectorization and optimization information to Cray FORTRAN programming. The following items have changes: nonrepeatable edit descriptors and the format specifications. The calling sequence information was moved from Appendix F to the Macros and Opdefs Reference Manual, CRI publication SR-0012. The console attention handler information was removed from Appendix I.
- J-01 July, 1983 - This change packet brings the manual into agreement with the CFT 1.11 release. Changing the default of IF optimization from OPT=PARTIALIFCON to OPT=NOIFCON on the CFT control statement is the only major change. Miscellaneous technical and editorial changes are also included.

J-02 January, 1984 - This change packet brings the manual into agreement with the CFT 1.13 release. The CFT release has been numbered 1.13 in conjunction with the 1.13 COS release. Major changes include the addition of: reentrancy support; new instruction scheduler; gather/scatter; dollar sign editing; the ALLOC, CPU, DEBUG, and SAVEALL control statement parameters; SAFEDOREP, FULLDOREP, NODOREP, INVMOV, NOINVMOV, UNSAFEIF, SAFEIF, BL, NOBL, BTREG, and NOBTREG control statement options; the U compiler option; UNSAFEIF, SAFEIF, BL, and NOBL scheduler directives; RESUMEDOREP and NODOREP optimization directives; implementing the ALIGN directive; DEBUG and NODEBUG directives; dependency information; population parity count Boolean function; and new CFT messages. The M and Y compiler options, the SCHED/NOSCH compiler directives, and several CFT messages have been removed. Miscellaneous technical and editorial changes are also included.

PREFACE

The Cray FORTRAN Compiler (CFT) translates FORTRAN language statements into Cray Assembly Language (CAL) programs that make effective use of the CRAY-1 and CRAY X-MP Computer Systems. This manual describes the Cray FORTRAN language in its entirety and related Cray Operating System (COS) characteristics.

This manual is a reference manual for CFT programmers. The programmer is assumed to have a working knowledge of the FORTRAN programming language. However, when basic terms and concepts are being defined, they are italicized.

This publication is divided into three parts as described below.

PART 1 - CFT DESCRIPTION

Sections 1, 2, and 3 describe the CFT language. The presentation progresses from the identities and forms of basic syntactic elements through the methods of data representation and the evaluation of expressions.

Section 4 describes functions and subroutines.

PART 2 - CFT STATEMENTS

Part 2 presents the CFT statements that conform to the ANSI standards and CFT statements that are an extension of those standards. CFT extensions give the programmer a broader range of capabilities. Extensions that can be conveniently replaced with standard statements are described in Appendix E.

PART 3 - THE CFT COMPILER

Part 3 presents the CFT control statement, directives that control compilation, and techniques for improving the performance of CFT programs.

CONTENTS

<u>PREFACE</u>	v
--------------------------	---

PART ONE - CFT DESCRIPTION

1. <u>CRAY FORTRAN LANGUAGE</u>	1-1
THE CRAY FORTRAN COMPILER	1-1
CONFORMANCE WITH THE ANSI STANDARD	1-1
CONVENTIONS	1-2
ELEMENTS OF THE CFT LANGUAGE	1-3
Character sets	1-3
FORTRAN character set	1-3
Auxiliary character set	1-4
Uppercase/lowercase conversion	1-5
Sequences	1-5
Syntactic items	1-5
Constants	1-5
Symbolic names	1-6
Statement labels	1-6
Keywords	1-7
Operators	1-7
Lists and list items	1-7
FORTRAN statements	1-8
Lines	1-8
Comment lines	1-9
Initial lines	1-9
Continuation lines	1-9
Terminal lines	1-10
Compiler directive lines	1-10
THE EXECUTABLE PROGRAM	1-10
Program units	1-10
The main program	1-12
The subprogram	1-12
Normal execution sequence	1-12
Order of statements and lines	1-13
2. <u>DATA REPRESENTATION AND STORAGE</u>	2-1
TYPES OF DATA	2-1
Data type of an array element	2-2

TYPES OF DATA (continued)	
Data type of a function	2-2
CONSTANTS	2-2
Integer constants	2-3
Real constants	2-4
Basic real constant	2-5
Constant followed by a real exponent	2-5
Nonzero real constant range	2-5
Double-precision constants	2-5
Constant followed by a double-precision exponent	2-6
Nonzero double-precision constant range	2-6
Complex constants	2-6
Nonzero complex constant range	2-7
Logical constants	2-7
Boolean (octal) constants	2-8
Character constants	2-8
VARIABLES	2-9
ARRAYS	2-10
Array declarators	2-10
Format of an array declarator	2-10
Kinds of array declarators	2-11
Actual array declarators	2-11
Dummy array declarators	2-11
Size of an array	2-12
Array element names	2-12
Array storage sequence	2-12.1
Array element order	2-13
Subscript values	2-15
Dummy and actual arrays	2-15
Adjustable arrays and adjustable dimensions	2-16
Use of array names	2-17
CHARACTER SUBSTRINGS	2-18
STORAGE AND ASSOCIATION	2-18
Storage sequences	2-19
Association of entities	2-19
Definition	2-21
Defined entities	2-21
Undefined entities	2-22
SYMBOLIC NAMES	2-23
Scope of symbolic names	2-24
Global entities	2-24
Local entities	2-24
Classes of symbolic names	2-25
Common blocks	2-25
External functions	2-25
Subroutines	2-26
The main program	2-26
Block data subprograms	2-26
Arrays	2-26
Variables	2-27
Constants	2-27

Classes of symbolic names (continued)	
Statement functions	2-28
Intrinsic functions	2-28
Dummy procedures	2-28
NAMELIST group name	2-29
 3. <u>EXPRESSIONS</u>	3-1
ARITHMETIC EXPRESSIONS	3-1
Arithmetic operators	3-2
Interpretation of arithmetic operators in expressions	3-2
Precedence of arithmetic operators	3-3
Arithmetic operands	3-3
Primaries	3-4
Factors	3-4
Terms	3-5
Arithmetic expressions	3-5
Data type of arithmetic expressions	3-6
Integer quotients	3-8
Type conversion	3-9
Type integer	3-9
Type real	3-9
Type double-precision	3-9
Type complex	3-10
Type Boolean	3-10
Evaluation of arithmetic expressions	3-10
CHARACTER EXPRESSIONS	3-11
Character expression evaluation	3-11
RELATIONAL EXPRESSIONS	3-12
Arithmetic relational expressions	3-12
Character relational expressions	3-13
LOGICAL EXPRESSIONS	3-14
Logical operators	3-14
Form and interpretation of logical expressions	3-14
Values of logical factors, terms, disjuncts, and expressions	3-16
BOOLEAN (MASKING) EXPRESSIONS	3-17
PRECEDENCE OF ALL OPERATORS	3-18
EVALUATION OF EXPRESSIONS	3-19
Order of evaluation of functions	3-19
PARENTHESES AND EXPRESSIONS	3-20
SUMMARY OF RULES OF INTERPRETATION	3-20
 4. <u>SUBROUTINE, FUNCTION, AND SPECIFICATION SUBPROGRAMS</u>	4-1
SPECIFICATION SUBPROGRAMS	4-1
Named common blocks	4-1
PROCEDURE SUBPROGRAMS	4-2
Subroutine subprograms	4-2

PROCEDURE SUBPROGRAMS (continued)	
Actual arguments	4-2
Subroutine subprogram restrictions	4-3
Function subprograms	4-3
Statement functions	4-3
Referencing statement functions	4-4
Statement function restriction	4-4
External functions	4-5
Referencing external functions	4-5
Execution of external function references	4-5
Actual arguments for external functions	4-5
Intrinsic functions	4-6
Referencing intrinsic functions	4-6
Intrinsic function restrictions	4-6
Utility procedures	4-6
Function subprogram restrictions	4-7
Execution of function references	4-8
Referencing functions	4-8
Non-FORTRAN subprograms	4-9
ARGUMENTS	4-9
Dummy arguments	4-9
Actual arguments	4-10
Association of dummy and actual arguments	4-10
Variables as dummy arguments	4-11
Arrays as dummy arguments	4-12
Procedures as dummy arguments	4-12
Restrictions on the association of entities	4-13
COMMON BLOCKS	4-14

FIGURES

1-1 Executable program	1-11
2-1 Array storage sequence	2-13
2-2 Array element arrangement and reference	2-14

TABLES

1-1 Special characters	1-4
1-2 Required order of lines and statements	1-14
2-1 Constant value representation	2-3
2-2 Logical constant representation	2-7
2-3 Subscript evaluation	2-16
3-1 Arithmetic operators	3-2
3-2 Interpretation of operators in expressions	3-2
3-3 Precedence of arithmetic operators	3-3
3-4 Arithmetic operand, expression, and result typing relationships	3-7
3-5 Type conversion in assignment statements	3-8
3-6 Relational operators	3-12
3-7 Logical operators	3-14
3-8 Precedence among all operators	3-18

PART TWO - CFT STATEMENTS

1.	<u>FORTRAN STATEMENTS</u>	1-1
2.	<u>DATA SPECIFICATION</u>	2-1
	DECLARATION AND INITIALIZATION	2-1
	PARAMETER statement	2-1
	DIMENSION statement	2-2
	POINTER statement (CFT extension)	2-3
	DATA statement	2-4
	DATA statement restrictions	2-6
	TYPE STATEMENTS	2-7
	INTEGER, REAL, DOUBLE PRECISION, COMPLEX, and LOGICAL type statements	2-7
	CHARACTER type statement	2-8
	IMPLICIT statement	2-9
	IMPLICIT NONE statement (CFT extension)	2-10
	ASSOCIATION STATEMENTS	2-11
	EQUIVALENCE statement	2-11
	Equivalence association	2-12
	Array names and array element names	2-12
	Restrictions on EQUIVALENCE statements	2-12
	COMMON statement	2-13
	Common block storage sequence	2-14
	Size of a common block	2-14
	Common association	2-15
	Differences between named common and blank common	2-15
	Restrictions on COMMON and EQUIVALENCE statements	2-15
	INTRINSIC STATEMENT	2-15
	SAVE STATEMENT	2-16
3.	<u>ASSIGNMENT STATEMENTS</u>	3-1
	ARITHMETIC ASSIGNMENT STATEMENT	3-1
	LOGICAL ASSIGNMENT STATEMENT	3-2
	CHARACTER ASSIGNMENT STATEMENT	3-2
	ASSIGN STATEMENT	3-3
4.	<u>PROGRAM CONTROL STATEMENTS</u>	4-1
	UNCONDITIONAL GO TO STATEMENT	4-1
	COMPUTED GO TO STATEMENT	4-2
	ASSIGNED GO TO STATEMENT	4-3
	ARITHMETIC IF STATEMENT	4-4
	LOGICAL IF STATEMENT	4-4

PROGRAM CONTROL STATEMENTS (continued)

CONDITIONAL BLOCK STATEMENTS	4-5
IF-block	4-5
Block IF statement	4-6
END IF statement	4-6
ELSE IF-block	4-6
ELSE IF statement	4-7
ELSE-block	4-7
ELSE statement	4-7
Conditional block statement execution	4-8
DO STATEMENT	4-8
Terminal statement	4-10
DO variable	4-10
Range of a DO-loop	4-10
Active and inactive DO-loops	4-11
Executing a DO statement	4-11
Loop control processing	4-13
Execution of the range	4-13
Terminal statement execution	4-13
Incrementation processing	4-13
Transfer into the range of a DO-loop	4-14
CONTINUE STATEMENT	4-14
STOP STATEMENT	4-15
PAUSE STATEMENT	4-15
END STATEMENT	4-16

5. INPUT/OUTPUT STATEMENTS 5-1

INPUT/OUTPUT RECORDS	5-1
Formatted records	5-1
Unformatted records	5-2
End-of-file (ENDFILE) records	5-2
End-of-data records	5-2
INPUT/OUTPUT FILES	5-2
RECORD AND FILE POSITIONS	5-3
DATASETS	5-3
INTERNAL RECORDS AND FILES	5-4
SEQUENTIAL ACCESS OPERATIONS	5-4
DIRECT ACCESS OPERATIONS	5-5
Dataset position before data transfer	5-5
Sequential access	5-5
Direct access	5-5
UNITS	5-6
IDENTIFIERS	5-6
Unit identifiers	5-6
Dataset identifiers	5-7
Format identifiers	5-7
READ, WRITE, AND PRINT STATEMENTS	5-8
Control information lists	5-9
Input/output lists	5-10

Input/output lists (continued)	
Input list items	5-10
Output list items	5-11
Implied-DO lists	5-11
DATA TRANSFER	5-12
Direction of data transfer	5-12
Identifying a unit	5-13
Establishing a format	5-13
Data transfer	5-13
Unformatted data transfer	5-14
Formatted data transfer	5-14
Dataset position after data transfer	5-15
BACKSPACE, ENDFILE, AND REWIND STATEMENTS	5-16
BACKSPACE statement	5-17
ENDFILE statement	5-17
REWIND statement	5-17
INQUIRE STATEMENTS	5-17
Inquiry by dataset name	5-18
Inquiry by unit	5-18
INQUIRE statement restrictions	5-20
OPEN STATEMENT	5-20
CLOSE STATEMENT	5-21
NAMelist STATEMENT (CFT EXTENSION)	5-23
NAMelist input	5-24
NAMelist input variables	5-25
NAMelist input processing	5-26
User control subroutines	5-26
NAMelist output	5-28
User control subroutines	5-28
BUFFER IN AND BUFFER OUT STATEMENTS (CFT EXTENSIONS)	5-29
The UNIT function	5-33
The LENGTH function	5-34
RESTRICTIONS ON INPUT/OUTPUT STATEMENTS	5-34
I/O ERROR RECOVERY	5-34
 6. <u>FORMAT SPECIFICATION</u>	 6-1
FORMAT STATEMENTS	6-1
Format of a format specification	6-2
EDIT DESCRIPTORS	6-3
Interaction between I/O lists and format specifications	6-5
Positioning by format control	6-7
Internal representation	6-7
Apostrophe and quotation mark editing	6-7
H editing	6-8
Positional editing (T, TL, TR, and X)	6-8
T, TL, and TR editing	6-9
X editing	6-9
Slash editing	6-10
Colon editing	6-10

Positional editing (continued)	
Dollar sign editing (CFT extension)	6-10.1
P editing	6-11
Numeric editing (BN, BZ, S, SP, SS, I, F, E, D, and G)	6-12
BN and BZ editing	6-13
S, SP, and SS editing	6-13
Integer editing	6-13
F editing	6-14
E editing	6-16
D (double-precision) editing	6-17
G editing	6-17
Complex editing	6-19
O (octal) editing (CFT extension)	6-19
Z (hexadecimal) editing (CFT extension)	6-20
L (logical) editing	6-20
A (alphanumeric) editing	6-21
R (right-justified) editing (CFT extension)	6-23
LIST-DIRECTED I/O	6-24
List-directed input	6-24
List-directed output	6-26

7. <u>PROGRAM UNIT SPECIFICATION</u>	7-1
PROGRAM STATEMENT	7-1
FUNCTION SUBPROGRAMS	7-2
Function reference	7-2
Function statement	7-2
Statement function definition statement	7-3
SUBROUTINE AND CALL STATEMENTS	7-5
Subroutine reference	7-5
Execution of a CALL statement	7-5
SUBROUTINE statement	7-6
RETURN STATEMENT	7-6
Execution of a RETURN statement	7-7
Alternate return	7-7
ENTRY STATEMENT	7-8
Referencing a procedure subprogram entry	7-9
Entry association in function subprograms	7-9
ENTRY statement restrictions	7-9
EXTERNAL STATEMENTS	7-10
BLOCK DATA STATEMENTS	7-11

FIGURES

4-1 IF-levels and blocks	4-9
--------------------------	-----

TABLES

5-1	Print control characters	5-15
5-2	Inquiry specifiers and their meanings	5-19
5-3	OPEN specifiers and their meanings	5-22
5-4	CLOSE specifiers and their meanings	5-23
6-1	Edit descriptors with data types	6-5

PART THREE - THE CFT COMPILER

1.	<u>CFT COMPILER I/O</u>	1-1
	THE CFT CONTROL STATEMENT	1-1
	ERROR MESSAGES DURING PROGRAM EXECUTION	1-5
	INPUT TO CFT	1-5
	OUTPUT FROM CFT	1-8
	Listable output	1-8
	Page header lines	1-8
	Source statement listings	1-8.1
	BLOCK BEGINS messages	1-9
	Table of statement numbers	1-9
	Table of names encountered	1-9
	Address field	1-10
	Name field	1-10
	Type field	1-10
	Main usage field	1-10
	Block field	1-11
	Table of block names and lengths in octal	1-11
	Table of external names	1-12
	Table of loops encountered	1-12
	Cross reference information	1-12.1
	Messages	1-12.1
	Program unit page table	1-12.1
	Compiler options	1-12.2
	Using compiler directive lines	1-13
	COMPILER DIRECTIVES	1-13
	Listable output control directives	1-14
	EJECT directive	1-14
	LIST directive	1-14
	NOLIST directive	1-15
	CODE directive	1-15
	NOCODE directive	1-15
	Vectorization control directives	1-16
	VECTOR directive	1-16
	NOVECTOR directive	1-17
	NORECURRENCE directive	1-18
	IVDEP directive	1-19

VFUNCTION directive	1-19
NEXTSCALAR directive	1-20
SHORTLOOP directive	1-20
Integer control directives (INT24, INT64)	1-20.1
Multiply/divide directives (FASTMD, SLOWMD)	1-21
Flow trace directives (FLOW/NOFLOW)	1-21
Flow trace enable/disable	1-22
FLODUMP routine	1-23
Options	1-23
SETPLIMQ	1-23
ARGPLIMQ	1-24
FLOWLIM	1-24
Scheduler directives	1-24
UNSAFEIF/SAFEIF directives	1-24.1
BL/NOBL directives	1-24.1
Dynamic common block directive (DYNAMIC)	1-25
Array bounds checking directive (BOUNDS)	1-25
BOUNDS options	1-25
Optimization directives	1-26
BLOCK directive	1-27
NO SIDE EFFECTS directive	1-27
ALIGN directive	1-28
NOIFCON directive	1-28
RESUMEIFCON directive	1-29
RESUMEDOREP directive	1-29
NODOREP DIRECTIVE	1-29
Debugging directives (DEBUG, NODEBUG)	1-30
EXTERNAL ROUTINES	1-30

2. <u>CRAY FORTRAN PROGRAMMING</u>	2-1
VECTORIZABLE DO-LOOPS	2-1
Qualifications for vectorization	2-1
Entity categories	2-2
Dependencies	2-4
Informative dependency messages	2-10
Vectorization with arrays	2-10
Using optimized routines	2-11
Use of optimized routines by CFT	2-12
Conditional statements	2-13
General guidelines for vectorization	2-16

TABLES

1-1 Effect of ALLOC, SAVEALL, and BTREG on variable allocation . .	1-4.4
1-2 Compiler options	1-6
1-3 External routines	1-30
2-1 Array A elements in vector and scalar modes	2-4
2-2 Dependency information combinations	2-5

APPENDIX SECTION

A.	<u>CHARACTER SET</u>	A-1
B.	<u>CRAY FORTRAN INTRINSIC FUNCTIONS</u>	B-1
C.	<u>CRAY FORTRAN UTILITY PROCEDURES</u>	C-1
D.	<u>CFT MESSAGES</u>	D-1
	COMPILE-TIME MESSAGES	D-2
	LOGFILE MESSAGES	D-19
	INFORMATIVE DEPENDENCY MESSAGES	D-21
E.	<u>OUTMODED FEATURES</u>	E-1
	HOLLERITH CONSTANTS	E-2
	HOLLERITH EXPRESSIONS	E-4
	Hollerith relational expressions	E-5
	HOLLERITH FORMAT SPECIFICATION	E-6
	TWO-BRANCH ARITHMETIC IF STATEMENTS	E-6
	INDIRECT LOGICAL IF STATEMENTS	E-6
	FORMATTED DATA ASSIGNMENT	E-7
	Encode and decode statements	E-7
	The ENCODE statement	E-8
	The DECODE statement	E-9
	EDIT DESCRIPTORS	E-9
	DOUBLE DECLARATION STATEMENTS	E-10
	DATA STATEMENT FEATURES	E-10
	PUNCH STATEMENT	E-11
	TYPE STATEMENT DATA LENGTH	E-11
	RANDOM INPUT/OUTPUT OPERATIONS	E-13
	Creating a dataset for random access.	E-13
	Dataset connection	E-13
	Positioning while connected for random access	
	(GETPOS/SETPOS)	E-13
	READMS/WRITMS routines	E-14
	Modifying a record under random access	E-15
	Extended range of a DO-loop	E-15
	Noncharacter arrays for format specification	E-15
	EOF, IEOF, and IOSTAT functions	E-16
F.	<u>CREATING NON-FORTRAN PROCEDURES</u>	F-1
G.	<u>SYMBOLIC DEBUG PACKAGE</u>	G-1
H.	<u>UNBLOCKED DATASETS</u>	H-1

I.	<u>REPRIEVE PROCESSING</u>		I-1
	REPRIEVE INITIATION		I-1
	REPRIEVE TERMINATION		I-2

TABLES

B-1	Generic and specific intrinsic function names		B-7
C-1	CFT utility procedures		C-2
E-1	Data length		E-12

INDEX

The form of a *complex constant* is an ordered pair of optionally signed real or integer constants separated by a comma and enclosed in parentheses. The first real constant of the pair is the real portion of the complex constant and the second is the imaginary portion.

Nonzero complex constant range

Nonzero complex constant components (where $C = C_{\text{real}} + iC_{\text{imag}}$) are represented in the Cray Computer Systems by two normalized, floating-point binary values ($C_{\text{real}}, C_{\text{imag}}$) in the following range.

$$2^{-8191} \leq |C_{\text{real}}|, |C_{\text{imag}}| \leq 2^{8191}$$

Each component contains a maximum of 48 significant binary digits of precision, approximating to the following decimal range with approximately 14 decimal digits of accuracy.

$$10^{-2466} < |C_{\text{real}}|, |C_{\text{imag}}| < 10^{2466}$$

The ANSI FORTRAN Standard does not specify a range of values for complex constant components.

LOGICAL CONSTANTS

Logical data can assume only the logical values true and false. Logical data occupies one storage unit in a storage sequence.

The forms, values, and internal representations of a *logical constant* are shown in table 2-2.

Table 2-2. Logical constant representation

Form	Value	Internal representation
.TRUE. or .T.	true	A negative value
.FALSE. or .F.	false	A zero or positive value

The ANSI FORTRAN Standard does not provide for the .T. or .F. form of the logical constant.

BOOLEAN (OCTAL) CONSTANTS

Boolean data is a set of binary zeros and ones that accounts for the content of each bit position in a single storage unit (64-bit Cray computer word).

The ANSI FORTRAN Standard does not provide for Boolean constants.

A *Boolean constant* contains 1 to 22 octal digits (0 through 7) followed by the letter B. When all 22 octal digits express a Boolean constant, their binary equivalents correspond with the content of every bit position in the storage unit (64-bit word). In this case, the leftmost octal digit can be a 0 or a 1 only, specifying the content of the leftmost bit position (bit 0). Each successive octal digit specifies the contents of the next three bit positions until the last octal digit specifies the contents of the rightmost three bit positions (bits 61, 62 and 63). When the Boolean constant consists of fewer than 22 octal digits, the constant is right-justified with zeros filling the leftmost bit positions.

Examples:

<u>Boolean constant</u>	<u>Internal representation (octal)</u>
1274653312572676113745B	1274653312572676113745
0B	0000000000000000000000
17777777777777777777B	17777777777777777777
77740B	0000000000000000077740
00776B	000000000000000000776

CHARACTER CONSTANTS

A *character constant* consists of any ASCII characters listed in Appendix A as being capable of internal representation.

The form of a character constant is an apostrophe followed by a nonempty string of characters followed by an apostrophe. An optional form of a character constant is a character string delimited by two quotation marks.

The lower and upper bounds of dimension declarators are arithmetic expressions that can contain constants, symbolic names of constants, functions, array elements, or variables. If the type of an entity is real, the entity is truncated to integer. Functions, array elements, and variables can be used only in adjustable array declarators. (See the description of kinds of array declarators, later in this section). If the lower bound is omitted, its value is assumed to be 1. The bound d_1 must be less than or equal to d_2 .

The ANSI FORTRAN Standard does not permit dimension declarators to be functions, array elements, or noninteger variables.

The number of dimension declarators specified in an array declarator indicates the number of dimensions in the array. The minimum number of dimensions is one and the maximum is seven.

When using FORMAT as an array name, note that CFT treats FORMAT statements as special cases to allow asterisk edit descriptors. The example

```
110  FORMAT(I3*I4)=(I5*I6)
```

is ambiguous. A statement is identified by CFT as a FORMAT statement if it has a statement label and begins with the characters FORMAT(. Therefore, an assignment of an array named FORMAT may not have a statement label as in the following example.

```
110  CONTINUE
      FORMAT(I3*I4)=(I5*I6)
```

Kinds of array declarators

An array declarator is either an actual array declarator or a dummy array declarator.

Actual array declarators - An *actual array declarator* is a constant array declarator having an array name that is not a dummy argument. An actual array declarator is permitted in a DIMENSION, COMMON, or type statement.

Dummy array declarators - A *dummy array declarator* can be a constant or an adjustable or assumed-size array declarator. An *adjustable array declarator* is an array declarator that contains one or more variables. An *assumed-size array declarator* is an array declarator in which the upper bound of the last dimension declarator is an asterisk.

Dummy array declarators appear only in function or subroutine subprograms. They are permitted in DIMENSION or type statements, but not in COMMON statements. An array name used in a dummy array declarator in a subprogram must also appear as an argument in its FUNCTION, SUBROUTINE, or ENTRY statement.

SIZE OF AN ARRAY

The size of an array is the number of elements in the array and is equal to the product of the sizes of all dimensions for that array.

CFT allows a maximum array size of 4,194,304 Cray computer words. The Cray Computer System being used, the memory required for other than the executable program and related data storage, and the executable program size might further restrict the array size.

The ANSI FORTRAN Standard does not specify a maximum for array size.

ARRAY ELEMENT NAMES

The format of an array element name is

$a(s[,s]...)$

where a is the array name,
 $(s[,s]...)$ is a subscript, and
 s is a subscript expression.

The number of subscript expressions should equal the number of dimension declarators in the array declarator. Fewer subscript expressions cause a warning message to be issued.

The ANSI FORTRAN Standard does not provide for fewer subscript expressions than declarators.

A subscript expression must yield an integer value when evaluated and can contain references to integer or Boolean constants, variables, functions, and array elements. The evaluation of the subscript expression must not alter the value of other expressions within the same statement.

CFT uses 24-bit A registers for subscript calculations. Overflow on intermediate values greater than $2^{23}-1$ is not detected and using very large values in subscript expressions can produce unpredictable results.

ARRAY STORAGE SEQUENCE

An array has a storage sequence defined by the storage sequence of its elements. The number of storage units (words) in an array is the product of the number of the elements in the array and the number of storage units required for each element.

- An external procedure name
- An alternate return specifier

An actual argument in a subroutine reference can be a dummy argument appearing in a dummy argument list within the subprogram containing the reference.

Subroutine subprogram restrictions

A subroutine subprogram can contain any statement except a BLOCK DATA, FUNCTION, PROGRAM, or a second SUBROUTINE statement.

The symbolic name of a subroutine or a subroutine entry is a global name and must not be the same as another global name or local name in the referencing program unit. The referencing program unit cannot use a subroutine or subroutine entry name as an external function or external function entry name.

In a subroutine subprogram, the symbolic name of a dummy argument is local and cannot appear in an EQUIVALENCE, PARAMETER, DATA, or COMMON statement. The symbolic name of a dummy argument can be the same as a common block name. A character dummy argument with a length specification of (*) must not appear as an operand for concatenation, except in a character assignment statement.

FUNCTION SUBPROGRAMS

A *function subprogram* is a sequence of executable code which can be referenced from a main program or a procedure subprogram. A function subprogram, unlike a subroutine subprogram, is referenced by the appearance of its identifier in certain types of statements. Function subprograms can be statement functions, external functions, or intrinsic functions.

Statement functions

A *statement function* is specified by a single statement similar in form to an arithmetic, logical, or character assignment statement. This statement function definition statement can only appear after the specification statements and before the first executable statement of the program unit in which it is referenced. Since it is not a part of the normal execution sequence, a statement function definition statement is classified as a nonexecutable statement.

Statement functions can be specified within a main program, a function subprogram, or a subroutine subprogram. A statement function can only be referenced from a statement within the same program unit containing its specification. (See part 2, section 7 for the format of a statement function definition statement.)

Referencing statement functions - A statement function is referenced by using its function reference as a primary in an expression.

Execution of a statement function reference results in the following actions.

- Evaluation of actual arguments that are expressions
- Association of actual arguments with corresponding dummy arguments
- Evaluation of the statement function expression

The resulting value is used in the expression containing the function reference.

The actual arguments must agree in order, number, and type with the corresponding dummy arguments. An actual argument can be any expression except a character expression involving concatenation of an operand with a length specification of (*), unless the operand is the symbolic name of a constant.

Statement function restrictions - A statement function can be referenced only in the program unit with the statement function definition statement.

A statement function definition statement can reference another statement function preceding the reference. The symbolic name identifying a statement function cannot appear as a symbolic name in any specification statement except a type statement (to specify the type of the function) or as a common block name in the same program unit.

An external function reference (see next subsection, External functions) in the expression of a statement function definition statement must not cause a dummy argument of the statement function to become undefined or redefined.

The symbolic name of a statement function is a local name and cannot be the same as another entity name in the program unit except a common block name. The symbolic name of a statement function cannot be an actual argument and cannot appear in an EXTERNAL statement.

A statement function definition statement in a function subprogram cannot reference that function subprogram.

Function subprogram restrictions

A function subprogram can contain any statement except a BLOCK DATA, SUBROUTINE, PROGRAM, or a second FUNCTION statement.

The symbolic name of an external function or external function entry is a global name and cannot be the same as another global name. In a referencing program unit, an external function or external function entry name cannot be used as the subroutine name in a CALL statement.

The symbolic name of a function specified by a FUNCTION statement cannot appear in another nonexecutable statement except for a type statement and must only appear as a variable in executable statements.

If the type of a function is specified in a FUNCTION statement, the function name cannot appear in a type statement. (Redundant type specifications are not allowed.)

In a function subprogram, the symbolic name of a dummy argument is local and cannot appear in an EQUIVALENCE, PARAMETER, SAVE, DATA, or COMMON statement except as a common block name.

A function specified by a subprogram can be referenced within another procedure subprogram or the main program of the executable program. A function subprogram cannot directly or indirectly reference itself.

The symbolic name of a function subprogram must appear as a variable name in the function subprogram. During every execution of the subprogram, this variable must become defined and, once defined, can be referenced or become redefined. The value of the function is the value of this variable when a RETURN or END statement is executed in the subprogram. The type of this value is implicit to the function name unless INTEGER, REAL, DOUBLE PRECISION, COMPLEX, CHARACTER, or LOGICAL is specified to cause it to be overridden.

A function subprogram can define one or more of its dummy arguments to return values in addition to the value of the function. However, this redefinition must not affect any entities referenced on the line referencing the function.

An actual argument in a function reference can be a dummy argument appearing in a dummy argument list within the subprogram containing the reference.

The result type of a statement function or an external function reference is the same as the function name type and is specified the same as variables and arrays. The result type of each intrinsic function is specified in Appendix B. Each argument type and the number of actual arguments specified in a function reference must agree with the (dummy) arguments defined in the specification of the referenced function.

If a function subprogram name is of type character, each entry name in the function subprogram must be of type character. If the function subprogram name or any entry in the subprogram has a length of (*) declared, all such entities must have a length of (*) declared; otherwise, all such entities must have a length specification of the same integer value.

A character dummy argument with a length specification of (*) cannot appear as an operand for concatenation, except in a character assignment statement.

Execution of function references

A function reference appears only as a primary in an arithmetic, character, or logical expression. Execution of a function reference in an expression causes the evaluation of the function identified by the symbolic name of the function subprogram.

Return of control from a referenced function completes execution of the function reference. The value of the function is then available to the expression containing the reference and being evaluated.

Referencing functions

A function is referenced in an expression and supplies a value to the expression. This value is the value of the function at the time the expression containing its reference is evaluated.

An intrinsic function can be referenced in the main program or in any procedure subprogram of an executable program.

A statement function can be referenced only in the program unit in which the statement function statement appears.

An external function can be referenced by function or entry name within another procedure subprogram or the main program of the executable program. A subprogram must not reference itself, either directly or indirectly.

Using the ENTRY statement, a procedure subprogram can be entered at any executable statement not within a DO-loop or block IF range. A procedure subprogram can contain one or more ENTRY statements following its FUNCTION or SUBROUTINE statement. (See part 2, section 7 for the format of the ENTRY statement.)

If a character function is referenced in a program unit, the function length specified in the program unit must be an integer constant expression.

Entities appearing in DATA statements are assigned to static storage.

The ANSI FORTRAN Standard does not specify storage allocation methods.

The format of a DATA statement is

DATA *nlist*/*clist*/[[,*nlist*/*clist*/]...

where *nlist* is a list of variable names, array names, array element names, substring names, and implied-DO lists separated by commas, and

clist is a list of the form

$[r^*]c[, [r^*]c]...$

where *c* is a constant or the symbolic name of a constant, and

r is a nonzero, unsigned, integer constant or the symbolic name of such a constant.

The r^*c form is interpreted to provide *r* successive values of the constant *c*.

The *i*th entity in *nlist* becomes defined with the *i*th value from *clist*.

An implied-DO list in a DATA statement has the format

(*dlist*, $i=e_1, e_2$ [, *e*₃])

where *dlist* is a list of array element names and implied-DO lists separated by commas,

i is the name of an integer variable called the *implied-DO variable*, and

*e*₁, *e*₂, and *e*₃ are integer expressions containing integer constants, the names of integer constants, and implied-DO variables of other implied-DO lists containing the implied-DO list within their ranges. If omitted, *e*₃ is assumed specified as 1.

The range of an implied-DO list is the list *dlist*. The iteration count and values of the implied-DO variable *i* are established the same as a DO-loop except the iteration count must be greater than zero. Interpretation of an implied-DO list in a DATA statement causes each item in the list *dlist* to be specified once for each iteration, and for appropriate values to be substituted where implied-DO variables are referenced.

Each subscript expression in the list *dlist* must be an integer constant expression, except the expression may contain implied-DO variables of implied-DO lists having the subscript expressions in their ranges.

In the following example, the first ten values of array A are set to 1.

```
DIMENSION A(25)
DATA (A(I),I=1,10)/10*1/
```

DATA statement restrictions

Names of constants, dummy arguments, functions, and entities in blank common (including entities associated with an entity in blank common) must not appear in *nlist*. Names of entities in a named common block can appear in *nlist*.

The same number of items must be specified by each *nlist* and its corresponding *clist*. The initial values of the entities are defined by this correspondence.

When the *nlist* entity is of type integer, real, or double-precision, the corresponding *clist* constant is converted, if necessary, to the type of the *nlist* entity according to the rules for arithmetic conversion. An *nlist* entity of type logical must correspond to a *clist* constant of the same type. A *clist* entity of type logical must correspond to an *nlist* entity of type logical.

An *nlist* entity corresponding to a character constant must be of type character. If the character entity length in the list *nlist* is greater than its corresponding character constant length, the additional rightmost characters in the entity are initially defined with blank characters. If the character entity length in the list *nlist* is less than its corresponding character constant length, the additional rightmost characters in the constant are ignored.

Any variable or array element can be initially defined except for the following.

- An entity that is a dummy argument
- An entity in blank common, which includes an entity associated with an entity in blank common

- A variable in a function subprogram whose name is also the name of the function subprogram

Subscript expressions in the list *nlist* must be integer constant expressions except for implied-DO variables. Substring expressions in the list *nlist* must be integer constant expressions.

- Intrinsic subroutine and function names need not appear in explicit type statements and must not be declared EXTERNAL.
- Pointers appearing in a POINTER statement are always assumed to be of type integer and, therefore, need not be explicitly typed as such.

ASSOCIATION STATEMENTS

Association statements specify the relationship of entities to storage units within the same program or among two or more programs.

EQUIVALENCE STATEMENT

An EQUIVALENCE statement specifies the sharing of one or more storage units by two or more entities in a single program unit. This causes the association of those entities.

If associated entities are of different data types, the EQUIVALENCE statement does not cause type conversion or imply mathematical equivalence. If a variable and an array are associated, the variable does not assume the properties of an array and the array does not assume the properties of a variable.

Associated entities are assigned to the same type of storage, static or stack storage. Associated entities are assigned to static storage unless the ALLOC=STACK option is specified on the CFT control statement (see part 3, section 1) and the entities have not been previously assigned to static storage (for example, with a DATA statement).

The ANSI FORTRAN Standard does not specify storage allocation methods.

The format of an EQUIVALENCE statement is

EQUIVALENCE (*nlist*) [, (*nlist*)]...

where *nlist* is a list of two or more variable names, array element names, character substring names, and array names, separated by commas.

Names of dummy arguments of a subprogram cannot appear in *nlist*. A variable name that is also a function name cannot appear in *nlist*.

Each subscript expression or substring expression in *nlist* must be an integer constant expression.

Equivalence association

An EQUIVALENCE statement specifies that the storage sequence of each entity in a list *nlist* shares the same first storage unit. This causes the association of all entities in the list and can also cause indirect association of other entities.

Array names and array element names

If an array element name appears in an EQUIVALENCE statement, the number of subscript expressions must equal the number of dimensions in the array declarator for the array.

The use of an array name in an EQUIVALENCE statement has the same effect as using the name of the first array element.

Restrictions on EQUIVALENCE statements

An EQUIVALENCE statement must not specify the same storage unit to occur more than once in a storage sequence. For example,

```
DIMENSION A(2)
```

```
EQUIVALENCE (A(1) ,B) , (A(2),B)
```

is prohibited because it would specify the same storage unit for A(1) and A(2).

An EQUIVALENCE statement must not specify consecutive storage units to be nonconsecutive. For example, the following is prohibited.

```
REAL A(2)
```

```
DOUBLE PRECISION D(2)
```

```
EQUIVALENCE (A(1),D(1)), (A(2),D(2))
```

An EQUIVALENCE statement must not associate the storage sequences of two different common blocks in the same program unit.

Example:

COMMON/A/X

COMMON/B/Y

EQUIVALENCE (X,Y)

EQUIVALENCE statement association must not cause extending of a common block storage sequence by adding storage units preceding the first storage unit of the first entity specified in a COMMON statement for the common block. For example,

The ANSI FORTRAN Standard does not provide for variable size for named common blocks.

Common association

In an executable program, common block storage sequences of all common blocks with the same name share the same first storage unit. The same is true of all blank common blocks. This associates entities with different program units.

Differences between named common and blank common

A blank common block has the same properties as a named common block except entities in blank common blocks cannot be initially defined by DATA statements.

Restrictions on COMMON and EQUIVALENCE statements

An EQUIVALENCE statement must not associate the storage sequences of two different common blocks in the same program unit. EQUIVALENCE statement association must not extend a common block storage sequence by adding storage units preceding the first storage unit of the first entity specified in a COMMON statement for the common block.

INTRINSIC STATEMENT

An INTRINSIC statement identifies a symbolic name as an intrinsic function. It permits use of a specific intrinsic function name as an actual argument.

The format of an INTRINSIC statement is

INTRINSIC <i>fun</i> [, <i>fun</i>]...

where *fun* is an intrinsic function name.

The appearance of a name in an INTRINSIC statement declares the name is an intrinsic function name. If an intrinsic function name is an actual argument in a program unit, it must appear in an INTRINSIC statement in the program unit.

The following intrinsic function names must not appear as actual arguments.

AMAX0	CHAR	DMIN1	IFIX	LLE	MAX1	SNGL
AMAX1	CMPLX	FLOAT	INT	LLT	MIN	REAL
AMIN0	DBLE	ICHAR	LGE	MAX	MIN0	
AMIN1	DMAX1	IDINT	LGT	MAX0	MIN1	

The appearance of a generic function name in an INTRINSIC statement does not cause loss of the name's generic property.

A given symbolic name must not appear in both an EXTERNAL and an INTRINSIC statement. In addition, it can appear only once in all of the INTRINSIC statements of a program unit. Appendix B lists the intrinsic functions.

SAVE STATEMENT

A SAVE statement retains the definition status of an entity after the execution of a RETURN or END statement in a subprogram. The entity remains defined in the current program unit only. The SAVE statement must appear before any executable statement in a program unit. All entities specified in a SAVE statement are assigned to static storage.

The ANSI FORTRAN Standard does not specify storage allocation methods.

The format of a SAVE statement is

SAVE [a[,a]...]

where a is a named common block name preceded and followed by a slash; a variable name; or an array name.

The names of dummy arguments, pointers, or procedures must not be specified in a SAVE statement. Variables and arrays in a common block must not be specified except by specifying the entire block. A common block specified in a SAVE statement must also be specified in every subprogram where the common block appears.

If a is omitted, all common blocks, variables, and arrays are assumed specified.

A name cannot appear more than once in the SAVE statements of a program unit.

A SAVE statement is optional in the main program and has no effect on the main program.

See the CFT control statement option BTREG in part 3, section 1 for more information on SAVE statement allocation.

Assignment statements define variables and array elements during program execution. Categories of assignment statements are as follows.

- Arithmetic
- Logical
- Character
- ASSIGN (statement label)

ARITHMETIC ASSIGNMENT STATEMENT

The format of an arithmetic assignment statement is

$$v = e$$

where v is the name of a variable or array element of type integer, real, double-precision, or complex, and e is an arithmetic expression.

Execution of an arithmetic assignment statement causes the evaluation of the expression e , conversion of e to the type of v (if required), and definition of v with the resulting value. Table 3-4 in part 1, section 3 relates such conversions to the data types of arithmetic operands, expressions, and evaluations.

Examples:

The statement ...

```
L = 12
C = (0.8,16.5) - (16.32,-6.1)
X = -B + (B**2-4*A*C)**0.5
A= B + L
ROOT = SQRT(65536.0)
ARRAY(6,2,1)=0
MATRIX(I,J,K)=MATRIX(I,J,K)+1
```

Assigns to v ...

```
Integer variable
Complex variable
Real variable
Real variable
Real variable
Real array element
Integer array element
```

LOGICAL ASSIGNMENT STATEMENT

The format of a logical assignment statement is

$$v = e$$

where v is the name of a logical variable or array element, and
 e is a logical expression.

Execution of a logical assignment statement causes the evaluation of the expression e and the definition of v with the value of e .

Examples:

All variable and array element names are assumed to be of type logical except for E and F, which are type real.

```
T = .FALSE.  
A = B  
C = (A .AND. B) .OR. (C .AND. D)  
T = .NOT. T  
TRUTAB(I,J,K,L) = .T.  
T = E.GE.F .OR. E/F .LT..4  
T = A .EQV. B
```

CHARACTER ASSIGNMENT STATEMENT

The format of a character assignment statement is

$$v = e$$

where v is the name of a variable, array element, or substring of type character, and
 e is a character expression.

Execution of a character assignment statement causes the evaluation of the expression e and the definition of v with the value of e . e is either truncated or padded with blanks on the right, as necessary, to match the length of v . No character positions defined in v can be referenced in e .

CONTROL INFORMATION LISTS

The format of a *control information list (cilst)* is

$$[\text{UNIT}=] \begin{Bmatrix} u \\ \text{din} \end{Bmatrix} [, [\text{FMT}=] f] [, \text{END}=sn] [, \text{REC}=rn] [, \text{ERR}=s] [, \text{IOSTAT}=ios]$$

where $[\text{UNIT}=]u$ is the unit specifier and $[\text{UNIT}=]\text{din}$ is the dataset specifier. The control information list must contain a unit specifier or a dataset specifier, but not both. If the UNIT= keyword is omitted, u and din are positional parameters and must appear first.

The ANSI FORTRAN Standard does not provide for the $[\text{UNIT}=]\text{din}$ form.

$[\text{FMT}=]f$ is the format specifier. This parameter must be present for formatted input/output statements. If f is an asterisk, the statement is list-directed and a record specifier cannot be present. If the optional UNIT= keyword is specified with the unit or dataset identifier, the FMT= keyword must be specified with the format identifier. If both the UNIT= and the FMT= keywords are omitted, f must follow u or din .

$\text{END}=sn$ indicates an end-of-file specifier. sn is the number of the statement where execution continues after an EOF on a READ statement has been encountered. An end-of-file specifier must not appear in a WRITE statement or in the same control information list as a record specifier.

$\text{REC}=rn$ indicates a record specifier. rn must be a positive integer expression with a positive value. A record specifier appears only in direct-access input/output statements. A statement containing a record specifier cannot contain an end-of-file specifier.

$\text{ERR}=s$ indicates an error specifier. s is the statement label of the statement where control continues after encountering a recoverable error.

IOSTAT=*ios*

is a status specifier that becomes defined when an input/output statement is executed. *ios* must be an integer variable or an integer array element. Following are the specifier values and their meanings.

<u><i>ios</i> value</u>	<u>Meaning</u>
=0	Transfer is complete; no error or end-of-file condition exists.
>0	Error message number; see coded \$IOLIB messages in CRAY-OS Message Manual, publication SR-0039.
<0	End of file was encountered; no error condition exists.

Examples:

READ (98,12345,ERR=42,END=75) ...	WRITE (17,25) ...
READ (10,REC=J) ...	READ (K+1) ...
READ (J,ARRAYF,ERR=10,END=25) ...	READ (98,ERR=37) ...
READ *,JOE...	READ (END=100,FMT=20,UNIT=5) ...
PRINT 22	READ (10,IOSTAT=JOE)
READ (*,*) JOE	READ (10,IOSTAT=JOE,ERR=100,END=200)
READ (10,*) JOE	
READ (98,'(6E11.4)',END=75)	

INPUT/OUTPUT LISTS

An *input/output list (iolist)*, specifies entities whose values are transferred by input/output statements. This list is composed of one or more input/output list items separated by commas. Optionally, one or more implied-DO lists can be included in the list.

An array name appearing as an input/output list item is treated as if all elements of the array were specified in the order given by array element ordering.

Input list items

Only input list items can appear in an input statement. An *input list item* must be one of the following.

- Variable name
- Array element name

BACKSPACE STATEMENT

A BACKSPACE statement causes the dataset related to the specified unit to be positioned at the beginning of the preceding record. If no preceding record exists, the position of the dataset is unchanged. If the preceding record is an endfile record, the dataset is positioned before it.

The ANSI FORTRAN Standard does not provide for backspacing a dataset that is not connected, a dataset that is connected but does not exist, or one that has been written with list-directed format.

ENDFILE STATEMENT

An ENDFILE statement writes an endfile record as the next record of the dataset. The dataset is then positioned after the endfile record.

After the execution of an ENDFILE statement, a BACKSPACE or REWIND statement must reposition the dataset before execution of an input statement. An output statement creates another file on the same dataset. Execution of an ENDFILE statement for a dataset that is connected but does not exist creates the dataset.

The ANSI FORTRAN Standard does not provide for the writing of an endfile on a dataset that is not connected.

REWIND STATEMENT

A REWIND statement causes the specified dataset to be positioned at its initial point. If the dataset is already positioned at its initial point, execution of the statement has no effect on the dataset position.

The ANSI FORTRAN Standard does not provide for the rewinding of an unconnected dataset or a dataset connected for direct access, and it does not provide for the creation of a connected dataset that did not exist.

INQUIRE STATEMENTS

An INQUIRE statement determines the current status of an external dataset's attribute. Inquiry can be made by dataset or by unit.

INQUIRY BY DATASET NAME

The format of the INQUIRE by dataset name statement is

`INQUIRE (idlist)`

where *idlist* is a list of inquiry specifiers that must contain exactly one dataset specifier and can contain, at most, one of each of the inquiry specifiers described in table 5-2.

■ The format of the dataset specifier is

`FILE=fin`

■ where *fin* is a character expression whose value specifies the name of the dataset. The named dataset need not exist or be connected to a unit. *fin* is limited to seven characters, not counting trailing blanks. If *fin* does contain trailing blanks, they are discarded.

INQUIRY BY UNIT

The format of the INQUIRE by unit statement is

`INQUIRE(iulist)`

where *iulist* is a list of inquiry specifiers that must contain exactly one external unit specifier and can contain, at most, one of each of the inquiry specifiers described in table 5-2.

The format of the external unit specifier is

`UNIT=u`

where *u* is an external unit identifier. (See unit identifiers described earlier in this section.) The unit specified need not exist or be connected to a dataset. If it is connected to a dataset, however, the inquiry includes the connected dataset.

If the dataset to be connected to the unit is not the same as the dataset where the unit is connected, the effect is as if a CLOSE statement without a STATUS= specifier had been executed for the unit immediately before the execution of the OPEN statement.

If the dataset to be connected to the unit is the same as the dataset where the unit is connected, only the BLANK= specifier can have a value that is different from the current value. Execution of the OPEN statement causes the new value of the BLANK= specifier to be in effect. The dataset position is unaffected.

If a dataset is connected to a unit, execution of an OPEN statement on that dataset and a different unit is not permitted.

CLOSE STATEMENT

A CLOSE statement terminates the connection of a particular dataset to a unit and rewinds the dataset. The format of a CLOSE statement is

<code>CLOSE (<i>el</i>list)</code>

where *el*list consists of an external unit specifier and, at most, one of each of the other specifiers described in table 5-4.

Execution of a CLOSE statement can occur in any executable program and need not occur in the same program unit as an OPEN statement.

A disconnected dataset or unit can be reconnected within the same executable program either to the same dataset or unit, or to a different dataset or unit, provided the dataset still exists. If the dataset is memory resident, CLOSE deletes the dataset regardless of the STATUS specifier.

The ANSI FORTRAN Standard provides an implicit CLOSE for all datasets upon normal program termination. CFT programs do not perform implicit CLOSEs; datasets do not rewind.

The ANSI FORTRAN Standard does not allow memory resident datasets which are automatically deleted regardless of the STATUS specifier.

Table 5-3. OPEN specifiers and their meanings

Specifier	Data Type	Meaning	Input (I) or Return value (RV)
UNIT= <i>u</i> [†]	Integer	External unit specifier	(I) Unit number
IOSTAT= <i>ios</i>	Integer variable or array element	Error status specifier	(RV) 0 if no error condition exists; error message number if error condition exists.
ERR= <i>s</i>	Statement label	Statement label where control is transferred if error condition exists	(I) FORTRAN statement label
FILE= <i>fin</i> ^{††}	Character expression	File specifier	(I) Name of dataset to be connected
STATUS= <i>sta</i>	Character expression	Disposition specifier (Default, 'UNKNOWN')	(I) 'OLD', dataset must exist and FILE= must be specified. 'NEW', dataset is created, status becomes 'OLD', FILE= must be specified. 'SCRATCH', dataset is deleted when CLOSE statement is executed or when program is terminated. Dataset must not be named. 'UNKNOWN', the status is 'SCRATCH' if no file specifier is supplied and the unit is not connected; otherwise, the status becomes 'OLD'.
ACCESS= <i>acc</i>	Character expression	Access specifier (Default, 'SEQUENTIAL')	(I) 'SEQUENTIAL' is access method; 'DIRECT' is access method.
FORM= <i>fm</i> ^{†††}	Character expression	Form specifier (Default, 'UNFORMATTED' if access is direct; 'FORMATTED' if access is sequential.)	(I) 'FORMATTED', formatted I/O; 'UNFORMATTED', unformatted I/O
RECL= <i>rl</i>	Positive integer expression	Record length for direct access method (omitted for sequential access)	(I) For formatted I/O, number of characters per record; For unformatted I/O, 8 times the number of words
BLANK= <i>blnk</i>	Character expression	Blank specifier (Default, 'NULL')	(I) 'NULL' if numeric input blanks are ignored; 'ZERO' if all nonleading blanks are treated as zeros. This specifier permitted on datasets opened for formatted I/O only.

[†] UNIT= does not need to be included in the unit specification if *u* is the first item in *olist*.

^{††} *fin* is limited to seven characters, not counting trailing blanks.

^{†††} CFT allows formatted and unformatted records in the same dataset (non-ANSI).

Table 5-4. CLOSE specifiers and their meanings

Specifier	Data Type	Meaning	Input (I) or Return value (RV)
UNIT= <i>u</i> [†]	Integer	External unit specifier	(I) Unit number
IOSTAT= <i>iost</i>	Integer variable or array element	Error status specifier	(RV) 0 if no error condition exists; error message number if error condition exists
ERR= <i>s</i>	Statement label	Statement label where control is transferred if error condition exists	(I) FORTRAN statement label
STATUS= <i>sta</i>	Character expression	Disposition specifier (Default, 'KEEP' if OPEN status is 'OLD', 'NEW', or 'UNKNOWN'. Default, 'DELETE' if OPEN status is 'SCRATCH' or dataset is memory resident.)	(I) 'KEEP', the dataset continues to exist after CLOSE statement execution. Do not specify 'KEEP' for a dataset with 'SCRATCH' status on an OPEN statement. 'DELETE', the dataset does not exist after execution of the CLOSE statement.

[†] UNIT= does not need to be included in the unit specification if *u* is the first item in *ellist*.

NAMelist STATEMENT (CFT EXTENSION)

A NAMelist statement provides a format-free method of specifying input/output lists.

The format of the NAMelist statement is

NAMelist/*group*/*v*[,*v*]...[[,*v*]/*group*/*v*[,*v*]...]...

where *group* is the group name for the following list, and

v is a variable name or an array name. *v* cannot be a dummy argument or a pointee variable.

The group name must be used only as a NAMelist group name within the program unit. It can be used in place of the FORMAT statement in the following I/O statements only. Every occurrence of a group name in NAMelist statements after the first occurrence is treated as a continuation of the first occurrence. Lists with the same group name are treated as a single group.

```

READ      (unit,group [,ERR=sn,END=sn])
WRITE     (unit,group [,ERR=sn])
READ      group
PRINT     group
PUNCH     group

```

Variable or array names are separated by commas in the NAMELIST statement. These names can be members of more than one NAMELIST group.

The NAMELIST statement must follow all declaratives affecting the variable or array names and must precede the first use of the group name in any I/O statement.

NAMELIST INPUT

An input NAMELIST group record consists of one or more physical records. Column 1 is ignored, except for a possible echo flag. The first nonblank character following column 1 must contain a NAMELIST delimiter (\$ or &), immediately followed by the group name and one or more blanks. The remaining portion of an input record contains as many variables desired with assigned values, separated by commas, in any order in one of the following forms.

variable=value

array=value[,value,...]

array(subscripts)=value[,value,...]

where *subscript* must be an integer constant; multiple array values are assigned in storage order.

Any value can be repeated by

*n*value*

where *n* is the repetition count.

An input NAMELIST physical record can contain up to 160 characters. Blanks can be used for readability but must not be embedded in names or values. Names or values cannot be continued from one physical record to another. A delimiter \$ or & terminates a group record. The next group record begins with the next delimiter.

An optional comment can appear between input NAMELIST group records. It can also appear within an input NAMELIST group record. A comment within the record must be preceded by a colon or semicolon. A comment, if included, is the last item in a physical record. An input NAMELIST group record can contain only comments, or can be entirely blank.

NAMELIST input variables

NAMELIST input variables can be of type integer, integer*2, real, double precision, complex or logical. If a type mismatch occurs across the equal sign, the value is converted to the declared type of the variable, following the rules of $v=e$ in part 2, section 3, except that conversions between complex and double precision, or logical and any other type are not allowed. Octal and hexadecimal constants are considered to be Boolean. Character constants can be assigned to noncharacter variables, where they are treated as Boolean. Character constants cannot be assigned to a complex or double-precision variable.

Integer, real, and double-precision values are specified in the normal FORTRAN manner. Octal constants are specified as *dddd...dB* or as *O'ddd...d[']*, where each *d* is a digit between 0 and 7. Hexadecimal constants are specified as *Z'hhh...h[']*, where each *h* is a hexadecimal character between 0 and 9, or between A and F. Up to 22 *d*'s or 16 *h*'s can be specified. If fewer than 22 or 16 are specified, the values are right-justified in the input word.

Logical values are specified as

.T[*string*], or
.F[*string*], or
T[*string*], or
F[*string*]

where *string* is an optional string of characters that does not contain the following characters.

Replacement	(=)
Delimiter	(\$ or &)
Separator	(,)
Comment	(: or ;)
Array name indicator	((

string is generally added for clarity. For example, T or .T can specify a logically true value, or, for clarity, .TRUE can be used.

Complex constants are represented as

(real,imag)

real and *imag* can be integer or floating-point constants.

NAMelist input processing

The NAMelist processor scans forward from the current position on the input dataset until it encounters a delimiter (\$ or &) as the first nonblank character immediately followed by the group name.

If end-of-file or end-of-data is encountered before the group name is located, the job either aborts or branches to the END= address.

If the processor finds a NAMelist record other than the one it is looking for, that record is skipped with an informative message to the logfile.

If the processor encounters an echo flag (E) in column 1 of any record, that record and all subsequent records processed by the current read are echoed to \$OUT.

The job aborts or the ERR= branch is taken if one or more of the following conditions exists.

- The record contains a variable name that is not in a NAMelist group.
- Punctuation is missing.
- The format of a constant is illegal.

User control subroutines

The following routines provide for control of the NAMelist input defaults. The mode setting indicates the action to be taken.

CALL RNLSKIP(<i>mode</i>)	Determines action taken if NAMelist sees a group name that is not the one being sought
<i>mode</i> > 0	Skips the record and issues a logfile message. (Default)
<i>mode</i> = 0	Skips the record
<i>mode</i> < 0	Aborts the job or goes to the optional ERR= branch
CALL RNLTYP(<i>mode</i>)	Determines action taken if a type mismatch occurs across the equal sign

<i>mode</i> $\neq$ 0	Converts the constant to the type of the variable. (Default)
<i>mode</i> = 0	Aborts the job or goes to the optional ERR= branch
CALL RNLECHO(<i>unit</i>)	Specifies output unit for error message and echo lines
<i>unit</i> < 0	Specifies that error messages go to \$OUT. Lines echoed because of an E in column 1 go to \$OUT. (Default)
<i>unit</i> $\geq$ 0	Specifies that error messages go to <i>unit</i> . All input lines are echoed on <i>unit</i> , regardless of any echo flags present. (<i>unit</i> =6 or 101 imply \$OUT.)

In the following user control subroutine argument lists, *char* is a character specified as *lLx* or *lRx*, and *mode* is a value which, if nonzero, adds the character to the set and which, if zero, removes the character from the set.

No checks are made to determine the reasonableness, usefulness, or consistency of the changes.

CALL RNLFLAG(<i>char,mode</i>)	Adds or removes <i>char</i> from the set of characters that, if found in column 1, initiates echoing of the input lines onto \$OUT. (<i>char</i> default is E.)
CALL RNLDELM(<i>char,mode</i>)	Adds or removes <i>char</i> from the set of characters that precede the NAMELIST group name and signal end of input. (<i>char</i> default is \$ or &.)
CALL RNLSEP(<i>char,mode</i>)	Adds or removes <i>char</i> from the set of characters that must follow each constant to act as a separator. (<i>char</i> default is ,.)
CALL RNLREP(<i>char,mode</i>)	Adds or removes <i>char</i> from the set of characters that occurs between the variable name and the value. (<i>char</i> default is =.)
CALL RNLCOMM(<i>char,mode</i>)	Adds or removes <i>char</i> from the set of characters that initiates trailing comments on a line. (<i>char</i> default is : or ;.)

NAMELIST OUTPUT

An output NAMELIST group record is written in the following general form.

```
& group name variable name = value,...,  
array name = value,...,value,...,&END
```

where *group name*, *variable name*, and *array name*
 are the names corresponding to the names in the
 NAMELIST statement.

For arrays, the values are listed in storage order and repeated values are listed as *n*value*.

Example:

```
&OUTPUT ARRAYX=3,7,4*5,2,&END
```

Logical values are listed as .T. or .F.

Example:

```
&OUTPUT LOGVAL=.T.,&END
```

Complex values are listed with real and imaginary portions, respectively.

Example:

```
&OUTPUT COMVAL=(2.5,3.),&END
```

An output NAMELIST group record can extend any number of lines (physical records). The first position of each line is normally blank. The first line contains the delimiter & in column 2, followed by the group name. The last line ends with the character string &END.

Default line width is 133 characters unless the unit is 102 (\$PUNCH), in which case the default line width is 80 characters. NAMELIST output is readable as NAMELIST input.

User control subroutines

The following routines provide the user control of the output format.

In the following subroutines, *char* can be any ASCII character specified by *lLx*, or *lRx*. No checks are made to determine if *char* is reasonable, useful, or consistent with other characters. If the default characters are changed, use of the output line as NAMELIST input might not be possible.

CALL WNLLONG(<i>length</i>)	Sets the output line length to <i>length</i> . <i>length</i> must be greater than 8 and less than 161. If <i>length</i> is too short for an actual output line, the job aborts. Setting <i>length</i> to -1 restores the default line length (80 for \$PUNCH; otherwise, 133).
CALL WNLDELM(<i>char</i>)	Changes the character preceding the group name and END from & to <i>char</i> .
CALL WNLSEP(<i>char</i>)	Changes the separator character immediately following each value from , to <i>char</i> .
CALL WNLREP(<i>char</i>)	Changes the replacement operator that comes between <i>name</i> and <i>value</i> , from = to <i>char</i> .
CALL WNLFLAG(<i>char</i>)	Changes the character written in column 1 of the first line from blank to <i>char</i> . Typically, <i>char</i> is used for carriage control if the output is to be listed, or for forcing echoing if the output is to be used as input for NAMELIST reads.
CALL WNLLINE(<i>value</i>)	Allows each namelist variable to begin on a new line. <i>value</i> = 0, no new line <i>value</i> = 1, new line for each variable

BUFFER IN AND BUFFER OUT STATEMENTS (CFT EXTENSIONS)

Buffered input/output operations initiate a transfer of data and allow the subsequent execution sequence to proceed concurrently with the actual transfer. Either the UNIT or LENGTH utility function must be referenced to cause a delay in an execution sequence, pending completion of a buffered input/output operation. These functions can also determine certain characteristics of that operation upon its termination. The amount of data to be transferred is specified in terms of Cray computer words with no consideration given to the type or format of information contained.

The following example is a sample program, an input listing, and an output listing, showing the use of the NAMELIST statement.

PROGRAM

```
PROGRAM EXAMPLE (TYPICAL NAMELIST I/O USAGE)
LOGICAL ALL DONE
REAL LENGTH
DATA DENSITY,LENGTH,WIDTH,HEIGHT,ALLDONE /4*1.0,.FALSE./
NAMELIST /INPUT/ LENGTH,WIDTH,HEIGHT,ALLDONE ,DENSITY
NAMELIST /OUTPUT/ WEIGHT,LENGTH,WIDTH,HEIGHT,DENSITY
10 READ INPUT
IF (ALLDONE) STOP
WEIGHT = DENSITY*LENGTH*WIDTH*HEIGHT
PRINT OUTPUT
GO TO 10
END
```

INPUT LISTING

```
INPUT DATA FOR PROGRAM EXAMPLE
NOTE THAT COMMENT CARDS MAY BE INTERSPERSED BETWEEN COMPLETE GROUPS
$INPUT $ USE DEFAULT VALUES
$INPUT LENGTH = 3.0, ; A LONG WIDE CASE
      WIDTH = 3. $
&INPUT DENSITY = .5 &END
&INPUT ALLDONE = TRUE &
/EOF
```

OUTPUT LISTING

```
&OUTPUT WEIGHT = 1., LENGTH = 1., WIDTH = 1., HEIGHT = 1., DENSITY = 1., &END
&OUTPUT WEIGHT = 9., LENGTH = 3., WIDTH = 3., HEIGHT = 1., DENSITY = 1., &END
&OUTPUT WEIGHT = 4.5, LENGTH = 3., WIDTH = 3., HEIGHT = 1., DENSITY = 0.5, &END
```


EDIT DESCRIPTORS

Edit descriptors specify the form of a record and direct the editing between characters in a record and their corresponding internal representation.

An edit descriptor is either a repeatable edit descriptor, *ed*, or a nonrepeatable edit descriptor, *ned*.

The formats of *repeatable edit descriptors* are

```
[r]Iw[.m]
[r]Fw.d
[r]Ew.d[Ee]
[r]Dw.d
[r]Dw.dEe (CFT EXTENSION)
[r]Gw.d[Ee]
[r]Ow (CFT EXTENSION)
[r]Zw (CFT EXTENSION)
[r]Lw
[r]A[w]
[r]Rw (CFT EXTENSION)
```

where I, F, E, D, G, O, Z, L, A, and R
indicate the manner of editing,

w, r, and e
are nonzero, unsigned integer constants, and

d and m
are unsigned integer constants.

The *repeat specification*, r, optionally precedes any repeatable edit descriptor.

Examples:

```
E20.5E3  3E20.5E3  I3  F8.5  E19.12  D8.1  G13.3
O23  Z6   L7   A8   R6   A   5I9   2F6.0  12E7.2  3D10.0
29G5.0  6O23  2Z10  7L7  3A5  4R4  3A
```

The formats of nonrepeatable edit descriptors are

```
'h1h2...hn' (apostrophe)
"h1h2...hn" (quotation mark) (CFT EXTENSION)
nHh1h2...hn
Tc
TLc
TRc
nX
X (CFT EXTENSION)
/ (slash)
n/ (slash) (CFT EXTENSION)
: (colon)
$ (dollar sign) (CFT EXTENSION)
kP
BN
BZ
S
SP
SS
```

where apostrophe, quotation mark, H, T, TL, TR, X, slash, colon, P, BN, BZ, S, SP, and SS indicate the manner of editing;

h is any ASCII character listed in Appendix A as capable of internal representation;

c and *n* are positive, nonzero, unsigned integer constants; and

k is an optionally signed integer constant.

Examples:

```
'AN APOSTROPHE EDIT DESCRIPTOR'
"A QUOTATION-MARK EDIT DESCRIPTOR"
T112
55X
/
6/
:
$
3P
TL3
TR6
BZ
SS
```

results in the printing of

```
X= 1234.56789 Y=.
```

Whereas execution of

```
PRINT 20,X  
20 FORMAT(' 'F10.5,:'Y= 'F10.5)
```

results in the printing of

```
X= 1234.56789.
```

DOLLAR SIGN EDITING (CFT EXTENSION)

The dollar sign character (\$) in a format specification modifies the carriage control specified by the first character of the record. In an output statement, the \$ descriptor suppresses the carriage return/line feed. In an input statement, the \$ descriptor is ignored. The \$ descriptor is intended primarily for interactive I/O; it leaves the terminal print position at the end of the text (instead of returning it to the left margin), so a typed response follows the output on the same line.

Example:

Execution of

```
        WRITE (6,100)
100    FORMAT('WHAT IS YOUR NAME?', $)
        READ (5,105)
105    FORMAT (4A8)
```

Results in the printing of

WHAT IS YOUR NAME?

The response (in this example, HARRY) can go on the same line

WHAT IS YOUR NAME? HARRY

P EDITING

A scale factor is specified by a P edit descriptor of the form kP , where k is an optionally signed integer constant called the *scale factor*. kP represents 10^k as a multiplier.

The scale factor is 0 at the beginning of each input/output statement. It applies to all subsequently interpreted F, E, D, and G edit descriptors until another scale factor is encountered and established. Note that reversion of format control does not affect the established scale factor.

The scale factor, k , affects editing in the following manner.

- With F, E, D, and G input editing (provided that no exponent exists in the field) and with F output editing, the scale factor causes the externally represented number to correspond to the internally represented number multiplied by 10 to the k th power.
- On input with F, E, D, and G editing, the scale factor has no effect if there is an exponent in the field.
- On output with E and D editing, the basic real constant part of the quantity to be produced is multiplied by the k th power of 10 and the exponent is reduced by k .
- On output with G editing, the effect of the scale factor is suspended unless the magnitude of the data to be edited requires the use of E editing. In this case, the scale factor has the same effect as with E output editing.

Examples:

Input field	9876.54	98.7654E2	9876.54	987.654	.864786D-4	86.4786E2
FORMAT statement	FORMAT (2PF8.3,	-2PE9.4,	F9.4,	0PG9.4,	D9.4,	-2PE9.4)
Internal representation	98.7654	9876.54	987654.	987.654	.0000864786	8647.86
Internal representation	9.87654	9876.54	9876.54	987.654	864.786	8647.86
FORMAT statement	FORMAT (2PF12.2,	-2PE12.4,	F12.4,	1PG12.2,	D12.4,	-2PE12.4)
Output field	987.65	.0099E+06	98.7654.	9.88E+02	8.6479D+02	.0086E+06

The scale factor k controls decimal normalization. If $-d < k < 0$, there are $|k|$ leading zeros and $d - |k|$ significant digits after the decimal point. If $0 < k < (d+2)$, there are k significant digits to the left of the decimal point and $d - k + 1$ significant digits to the right of the decimal point. Other values of k are not permitted.

NUMERIC EDITING (BN, BZ, S, SP, SS, I, F, E, D, AND G)

Numeric editing specifies input/output editing of integer, real, double-precision, and complex data. The following general rules apply.

- On input, leading blanks are not significant. Plus signs can be omitted. A field of all blanks has the value 0.
- On input with F, E, D, and G editing, a decimal point appearing in the input field overrides that portion of an edit descriptor specifying the decimal point location. The input field can have more digits than are used in approximating the value of the data. The excess digits are used to round to the approximation but are otherwise discarded.
- On output, a positive or zero internal value in the field is prefixed with blank characters except as described below for S, SP, and SS editing. A negative internal value in the field is prefixed with blank characters followed by a minus sign.
- On output, the representation is right-justified in the field.

If the number of characters produced by the editing is smaller than the field width, leading blanks are inserted in the field.

<u>Internal representation</u>	<u>G edit descriptor</u>	<u>Output field positions</u>											
		1	2	3	4	5	6	7	8	9	10	11	12
-324.876	G12.6	-	3	2	4	.	8	7	6				
.487295343397	G10.5	.	4	8	7	3	0						
-72.59	G10.3	-	7	2	.	6							
.000000000019	G12.2						.	1	9	E	-	1	0
.000000000019	G9.1				.	2	E	-	1	0			
10000.	G12.2						.	1	0	E	+	0	5
10000.01	G12.2						.	1	0	E	+	0	5
10000.	G12.2E1						.	1	0	E	+	5	
10000.	G12.2E4			.	1	0	E	+	0	0	0	5	

COMPLEX EDITING

Complex data consists of a pair of separate real data. Data editing must be specified by two successively interpreted A, D, E, F, G, O, R, or Z edit descriptors. The first of the edit descriptors specifies editing for the real part; the second for the imaginary part. The two edit descriptors can differ. Nonrepeatable edit descriptors can appear between two successive A, D, E, F, G, O, R, or Z edit descriptors.

O (OCTAL) EDITING (CFT EXTENSION)

The Ow edit descriptor indicates the processing of an input list item of type integer, real, complex, Boolean, or logical and a field width of w positions. A double-precision list item can be used with an Ow descriptor for output only.

On input, the field contains a string of from 0 to 22 octal digits or blanks, representing a binary value to be stored into the list item. This value is right-justified in the list item if fewer than 22 octal digits are contained in the field. Unspecified bit positions are cleared to 0. A blank field is considered to be a field containing all zeros. If the first nonblank character in the field is a minus, the ones complement of the value is stored.

On output, the internal representation of the list item is converted to octal and the rightmost w octal digits are right-justified in the field.

If the list item is not of type double-precision and the field is larger than 22 positions, the output contains leading blank characters. If the list item is of type double-precision and w is greater than 45, the output contains leading blank characters. If w is greater than 22, a blank character occupies position $(w-22)$ in the output field. This character indicates the beginning of the double-precision portion. To completely output a double-precision value, the value of w must be at least 45.

Z (HEXADECIMAL) EDITING (CFT EXTENSION)

The Zw edit descriptor indicates processing of a list item of type integer, real, complex, Boolean, or logical and a field width of w positions.

On input, the field contains a string of from 0 to 16 hexadecimal characters representing a zero or positive integral value (in the base-16 number system) to be stored into the list item. This value is right-justified in the list item if fewer than 16 hexadecimal characters are contained in the field; leading zeros are assumed. A blank field is assumed to be a field of all zeros. If the first nonblank character in the field is a minus, the ones complement of the value is stored.

On output, the internal representation of the list item is converted to a zero or positive hexadecimal value and the rightmost w digits are right-justified in the field. If the field is larger than 16 positions, leading blank characters are output.

L (LOGICAL) EDITING

The Lw edit descriptor indicates processing of a logical list item and an input or output field width of w positions. The specified input/output list item must be of type logical. On input, the list item becomes defined with logical data. On output, the list item must be defined with logical data.

The input field consists of a T for true or an F for false, optionally followed by additional characters. The field can contain a leading period or leading blanks.

The output field consists of $w-1$ blanks followed by a T or F, depending on the value of the internal data.

Type real or double-precision list items must be numeric and suitable for F editing.

A type complex list item consists of an ordered pair of numeric fields separated by a comma and enclosed in parentheses. The first numeric field is the real portion of the complex constant; the second numeric field is the imaginary portion. The end of a record can occur between the real portion and the comma or between the comma and the imaginary portion. Each numeric field can be preceded or followed by blanks.

A list item of type logical must not include either slashes or commas among the optional characters permitted for L editing.

A type character list item has an input form with a nonempty string of characters enclosed in apostrophes. Each apostrophe in a character constant must be represented by two consecutive apostrophes without a blank or end-of-record. Character constants can be continued from the end of one record to the beginning of the next record. The end of the record does not cause a blank or any other character to become part of the constant. The constant can be continued on to as many records as needed. A blank, comma, and slash can appear in character constants.

For example, if len is the list item length, w is the character constant length and len is less than or equal to w , the leftmost len characters of the constant are transmitted to the list item. If len is greater than w , the constant is transmitted to the leftmost w characters of the list item and the remaining $len-w$ characters of the list item are filled with blanks. The effect is as if the constant were assigned to the list item in a character assignment statement.

A null value has no characters before or between value separators. A null value has no effect on the definition status of the corresponding input list item. A single null value can represent an entire complex constant but it cannot be used as either the imaginary or the real portion alone. The end of a record following any other separator, with or without separating blanks, does not specify a null value.

A slash encountered as a value separator during execution of a list-directed input statement terminates execution of that input statement after the assignment of the previous value. If additional items are present in the input list, the effect is as if null values had been supplied for them.

All blanks in a list-directed input record are considered to be part of some value separator except for the following.

- Embedded blanks surrounding the real or imaginary portion of a complex constant
- Leading blanks in the first record read, unless immediately followed by a slash or comma

LIST-DIRECTED OUTPUT

The form of the values produced is the same as that required for input, except as noted otherwise. The values are separated by one of the following.

- One or more blanks
- A comma optionally preceded and followed by one or more blanks

New records begin as necessary but, except for complex and character constants, the end of a record does not occur within a constant and blanks do not appear within a constant.

Logical output constants are T for the value true and F for the value false.

Integer output constants are produced with the effect of an Iw edit descriptor, for some value of w.

Real and double-precision constants are produced with the effect of either an F edit descriptor or an E edit descriptor, depending on the magnitude x of the value and a range $10^{-2466} < x < 10^{2466}$. If the magnitude x is within this range, the constant is produced with $0PFw.d$; otherwise, $1PEw.dEe$ is used. Reasonable values of w , d , and e are used for each of the cases involved.

Complex constants are enclosed in parentheses, with a comma separating the real and imaginary portions. If two or more successive values in an output record have identical values, a repeated constant of the form $n*c$ is produced instead of the sequence of identical values.

Character constants are not delimited by apostrophes and are not preceded or followed by a value separator. Each internal apostrophe in a character constant is represented externally by one apostrophe. Character constants have a blank character inserted by the processor for carriage control at the beginning of any record that begins with the continuation of a character constant from the preceding record.

Slashes as value separators and null values are not produced by list-directed formatting.

Each output record begins with a blank character for carriage control when the record is printed.

The Cray FORTRAN Compiler (CFT) transforms a Cray FORTRAN language program into an executable program in relocatable binary that can be loaded and executed on the CRAY-1 and CRAY X-MP Computer Systems.

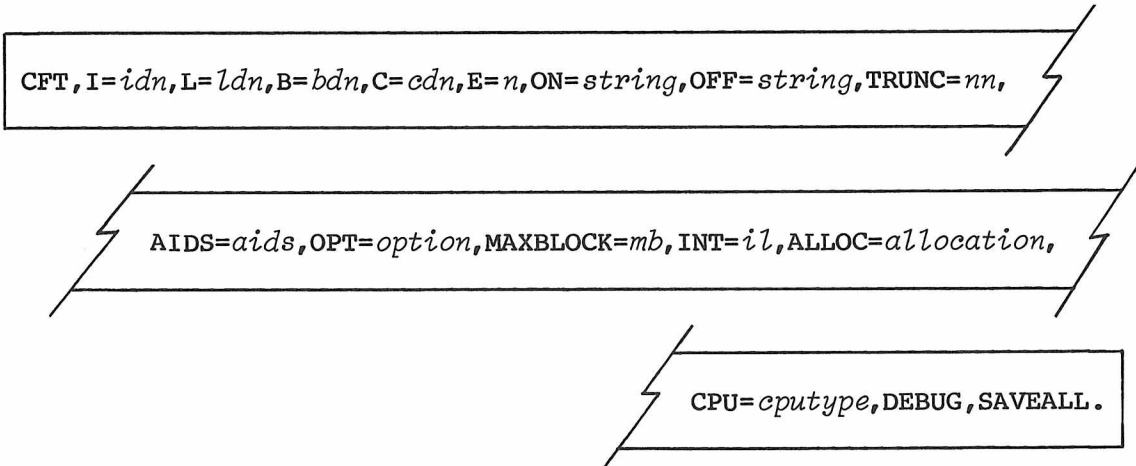
When a CFT control statement is executed, the Cray Operating System (COS) calls upon the system loader to load CFT from the mass storage subsystem. The compiler responds to information in a COS job deck to locate and compile the FORTRAN program. Both binary and symbolic information are output from the compiler.

In this section, CFT is described in terms of its input and output characteristics.

THE CFT CONTROL STATEMENT

The CFT compiler is loaded and executed when a CFT control statement is encountered in the control statement file of the job deck on the \$IN dataset.

The format of the CFT control statement is



CFT, I=*idn*, L=*ldn*, B=*bdn*, C=*cdn*, E=*n*, ON=*string*, OFF=*string*, TRUNC=*nn*,

AIDS=*aids*, OPT=*option*, MAXBLOCK=*mb*, INT=*il*, ALLOC=*allocation*,

CPU=*cputype*, DEBUG, SAVEALL.

Options can be in any order. If a keyword and option are omitted from the statement, the compiler uses a default value. A left parenthesis can be used in place of the first comma. A right parenthesis can be used in place of the period. If all options are omitted, a period can be used in place of empty parentheses.

The compiler does not reposition datasets before or after compilation.

- I=idn* Name of dataset[†] containing source input; default is \$IN.
- L=ldn* Name of dataset[†] to receive list output; default is \$OUT. L=0 suppresses all list output except for error messages written on \$OUT. If L=0 is specified, individual list options (see table 1-2) can be enabled by the ON= specification.
- B=bdn* Name of dataset[†] on which compiler writes binary load modules; default is \$BLD. If B=0, no binary load files are written. An end of file is not written.
- C=cdn* Name of pseudo Cray Assembly Language (CAL) dataset[†]; default is no dataset. This option provides for the generation of a text file that contains acceptable input to the CAL assembler with minor manual corrections. DATA statements are not supported with this option. It is intended to be used for hand coding of inner loops for enhanced efficiency.
- E=n* Level of severity of CFT-produced messages to be listed. The following levels are available.

<u>Message level</u>	<u>Severity type</u>	<u>Description</u>
1	COMMENT	Comments on programming inefficiencies (vectorization messages are controlled by the AIDS parameter)
2	NOTE	May cause problems with other compilers (Example: non-ANSI 66)
3	CAUTION	Possible user error (Example: no path to a statement)
4	WARNING	Probable user error (Example: using an array with too few subscripts)

[†] Dataset names are limited to seven characters.

<u>Message level</u>	<u>Severity type</u>	<u>Description</u>
5	ERROR	Fatal error
<i>n</i>		The highest message levels to be suppressed. For example, E=2 allows CAUTION, WARNING, and ERROR messages to appear. Fatal errors are never suppressed. Default is E=3. If E=0 is specified, no suppression takes place.

ON=*string* Enables compile options (see table 1-2, list of up to 15 characters representing options to be enabled)

OFF=*string*
Disables list or compile options (see table 1-2, list of up to 15 characters representing options to be disabled)

TRUNC=*nn* Number of bits to be truncated. Range is $0 \leq nn < 47$. Default is $nn=0$. Specifies truncation for all floating-point results. Does not truncate double-precision results, function results, or constants. Truncated bits are set to 0.

AIDS=*aids* Controls number of vectorization inhibition messages to be listed. *aids* can be one of the following.

<u><i>aids</i></u>	<u>Description</u>
LOOPNONE	No messages issued
LOOPPART	Maximum of 3 messages per inner DO-loop up to a total of 100 messages per compilation (default option)
LOOPALL	All messages issued

OPT=*option*
Specifies optimization options. When selecting multiple options, separate values by colons. Option values are:

<u><i>option</i></u>	<u>Description</u>
NOZEROINC	Assumes constant increment integers are not incremented by variables with the value 0 (default option)

<u>option</u>	<u>Description</u>
ZEROINC	Assumes constant increment integers (CII) can be incremented by variables with the value 0. This option inhibits the vectorization of any DO-loop in which there are CII's of the form $CII = CII + VARIABLE$.
NOIFCON	Disables optimization of conditional replacement statements of the form $IF(logical\ exp)\ var = expression$ except where CFT replaces these statements with MAX/MIN intrinsic functions (default option)
PARTIALIFCON	Allows CFT to optimize conditional replacement statements of the form $IF(logical\ exp)\ var = expression$ if var is of type integer, real, or logical, and $expression$ does not involve division or an external function reference. The optimization causes CFT to generate code similar to $var = CVMGX(expression, var, condition)$. If the optimization is performed, the IF statement will not inhibit vectorization or break an optimization block. See CDIR\$ NOIFCON and CDIR\$ RESUMEIFCON.
FULLIFCON	Allows CFT to optimize conditional replacement statements as described for PARTIALIFCON, except conditional replacement statements involving division and external functions are also optimized.
FASTMD	Causes CFT to use the fast integer multiply and divide algorithms. Operands and results are limited to 46 bits; there is no overflow protection.
SLOWMD	Causes CFT to generate the full 64-bit integer multiply and divide (default option)
SAFEDOREP	Enables replacement of 1-line DO-loops with a call to a \$SCILIB routine performing the same operation more efficiently (default option). Replacement does not occur when a 1-line DO-loop contains potential dependencies or equivalenced variables. See part 3, section 2 for examples on the use of SAFEDOREP. DO-loop replacement can be disabled and re-enabled within a program unit by specifying CDIR\$ NODOREP and RESUMEDOREP, respectively.

<u>option</u>	<u>Description</u>
FULLDOREP	Enables replacement of 1-line DO-loops with a call to a \$SCILIB routine performing the same operation more efficiently. Potential dependencies and equivalences are ignored. DO-loop replacement can be disabled and re-enabled within a program unit by specifying CDIR\$ NODOREP and RESUMEDOREP, respectively.
NODOREP	Disables replacement of 1-line DO-loops with a call to a \$SCILIB routine. NODOREP has no effect on vectorization of loops in the program. When OPT=NODOREP is specified, CDIR\$ RESUMEDOREP is ignored.
INVMOV	Enables movement of invariant code from a DO-loop body into the loop preamble (default option)
NOINVMOV	Disables movement of any invariant code from the DO-loop body into the loop preamble
UNSAFEIF	Enables instructions to move over a branch instruction by the instruction scheduler
SAFEIF	Disables instructions moving over a branch instruction. Prevents movement of a floating-point operation or subscripted reference before the branch of an IF statement put in to protect the operation (default option).
BL	Enables scalar loops to be bottom loaded; operand prefetched over the branch of the loop (default option).
NOBL	Disables scalar loops to be bottom loaded; intended to obtain correct code where the subscript for a load would be out of range if executed.
BTREG	Causes CFT to allocate specific scalar variables in a program unit to T registers during the program unit existence. Dummy arguments, arrays, and variables named in SAVE, DATA, COMMON, or NAMELIST statements are allocated to memory.

<u>option</u>	<u>Description</u>
BTREG (continued)	The maximum number of T registers available for variable allocation is 24. If there are fewer than 24 local integer (including INT24), real, logical, and compiler-generated variables, the remaining T registers are used as scratch registers during expression evaluation. If there are more than 24 variables in a program unit, the first 24 variables in the source code are allocated to the T registers and the remaining variables are allocated to memory. Specific variables can be forced into T registers by declaring them part of the first 24 variables at the beginning of a program unit. Variables can be excluded from T registers by specifying their names in a SAVE statement. Variables allocated to T registers are not initialized upon routine entry and become undefined when a RETURN or END is executed. Subprograms depending on local variables retaining their values across calls, which violates the ANSI FORTRAN standard, do not work properly unless the SAVE statement is used. (See the SAVEALL control statement option and table 1-1 in this section.) OPT=BTREG is the default when ALLOC=STACK is specified.
NOBTREG	Causes CFT to allocate all user variables to memory. NOBTREG does not affect the allocation of compiler-generated variables to B or T registers or the use of B or T registers temporarily holding values during expression evaluation. OPT=NOBTREG is the default unless ALLOC=STACK is specified.

MAXBLOCK=*mb*

Allows CFT to optimize or vectorize a block of code with a length up to *mb* words. Default is 2560 words of internal intermediate text. Values larger than 2560 may increase optimization but there may also be internal compiler errors (the errors may be undetected by CFT). MAXBLOCK=1 eliminates optimization or vectorization.

INT=*il* Length of integers. *il* values are as follows.

<u><i>il</i></u>	<u>Description</u>
------------------	--------------------

64	Full 64-bit integers (default option)
----	---------------------------------------

24	Short 24-bit integers
----	-----------------------

ALLOC=*allocation*

Specifies memory allocation scheme for entities in memory.
allocation can be one of the following.

<u><i>allocation</i></u>	<u>Description</u>
--------------------------	--------------------

STATIC	All memory is statically allocated; a stack is not used (default option). (See the SAVEALL parameter, BTREG control statement option, and table 1-1.)
--------	---

STACK	Read-only constants and entities in a DATA statement, SAVE statement, or a common block are statically allocated. All other entities are allocated on the stack.
-------	--

HEAP	Deferred implementation
------	-------------------------

CPU=*cputype*

Specifies mainframe type running the generated code;
default is the machine running CFT.

<u><i>cputype</i></u>	<u>Description</u>
-----------------------	--------------------

CRAY-1M	Generates code for CRAY-1 M Computer Systems
---------	--

CRAY-1S	Generates code for CRAY-1 S Computer Systems
---------	--

CRAY-XMP	Generates code for the CRAY X-MP Computer Systems
----------	---

DEBUG Writes sequence number labels at each executable FORTRAN statement to the Debug Symbol Table, allowing breakpoints to be set with SID at statement sequence numbers. DEBUG forces ON=IZ and sets MAXBLOCK=1. DEBUG on the control statement enables recognition of CDIR\$ DEBUG and CDIR\$ NODEBUG.

If DEBUG is not specified on the control statement (default), CDIR\$ DEBUG and CDIR\$ NODEBUG are ignored and debugging is turned off for the compilation.

SAVEALL Compilation occurs as if a SAVE statement with an empty list was in each program unit. All user variables in a program unit are allocated to static storage. Compiler-generated variables are allocated to B or T registers.

SAVEALL overrides OPT=BTREG. SAVEALL can be specified with ALLOC=STACK, that is, CFT uses the stack only for compiler-generated variables, argument lists, etc. (See the BTREG option, the ALLOC parameter, and table 1-1.)

Table 1-1. Effect of ALLOC, SAVEALL, and BTREG on variable allocation

When options are:			Variable appears in:			
			SAVE, DATA, or COMMON statement [†]	Array declaratives: CHARACTER COMPLEX DOUBLE EQUIVALENCE NAMELIST [†]	Other user variables ^{††}	Compiler-generated variables ^{†††}
ALLOC=	SAVEALL specified	BTREG or NOBTREG				
STATIC (default)	no (default)	NOBTREG (default)	static	static	static	register or static
STATIC (default)	no (default)	BTREG	static	static	register or static	register or static
STATIC (default)	yes	either	static	static	static	register or static
STACK	no (default)	NOBTREG	static	stack	stack	register or stack
STACK	no (default)	BTREG (default)	static	stack	register or stack	register or stack
STACK	yes	either	static	static	static	register or stack

[†] An entity appearing in (or equivalenced to an entity appearing in) a SAVE, DATA, or COMMON statement is allocated to static memory; otherwise, refer to the array declaratives or other user variable entries in table 1-1.

^{††} A user variable is allocated to a T register if it appears in table 1-1 and the variable is one of the first 24 variables in the program unit. Local scalar variables not appearing in EQUIVALENCE, DATA, or NAMELIST declaratives are also allocated to T registers.

^{†††} Compiler-generated variables include DO-loop trip counts, dummy argument addresses, temporaries used in expression evaluation, argument lists, and variables storing adjustable dimension bounds at entries. A compiler-generated variable is allocated to a register or memory depending on how the variable is used.

where

register	Value allocated to the T register or value or address allocated to the B register (memory is not allocated); values terminated when a END or RETURN is executed.
static	Value allocated to Permanent Static Memory maintained as long as the job step exists.
stack	Value allocated to Local Stack Memory; values terminated when an END or RETURN is executed.

ERROR MESSAGES DURING PROGRAM EXECUTION

While under control of COS, the executable program calls on system routines to accomplish its mathematical, input/output, and utility operations. These operations are required during compilation of the program. They are loaded from the system or user libraries and linked to the program by the system loader (LDR). When used, the routines respond to programming and/or equipment discrepancies by placing messages in the jobfile and in the \$OUT dataset. These discrepancies also cause the job to abort. The COS error messages and descriptions appear in the CRAY-OS Message Manual, CRI publication SR-0039.

INPUT TO CFT

CFT, when initiated, seeks two types of information: the program to be compiled and instructions on controlling the compilation.

A FORTRAN program to be compiled by CFT must be specified in a form using the ASCII character codes listed in Appendix A and the formats specified in part 2. The result is *source code*.

Other information required by CFT to complete its operations is provided by COS and compiler directives specified in the program being compiled. (See a description of compiler directives later in this section.) This information includes identification of the input dataset containing the source and identification of datasets receiving binary and listable output from CFT during compilation. The CFT options to use are specified in the CFT control statement. (See the subsection on the CFT control statement described earlier in this section.)

Table 1-2. Compiler options

Option	Description	Default
A	Aborts job after compilation if any program unit contains a fatal error	OFF
B	Lists beginning sequence number of each code generation block (G implies B)	OFF
C	Lists common block names and lengths listed on <i>ldn</i> after each program unit	ON
D	Lists DO-Loop Table	OFF
E	Enables recognition of compiler directive lines	ON
F	Enables FLOWTRACE option. (Also see FLOW/NOFLOW directives)	OFF
G	Lists generated code for each program unit. [†] (See CODE/NOCODE directives)	OFF
H	Causes listing of the first statement of each program unit and error messages. All other list options are ignored or disabled.	OFF
I	Enters compiler-generated statement labels in the Symbol Table	OFF
J	Causes all DO-loops to be executed at least once	OFF
L	Enables recognition of output listing control directives	ON
N	Enters null symbols in the Symbol Table (defined but not referenced)	OFF

[†] The G option lists the skeleton for the code generated for ENTRY and RETURN sequences. The actual number of B and T registers saved and the address where they are saved are not indicated. If no T registers are to be saved, the instruction to save T registers is replaced by a pass instruction.

Table 1-2. Compiler options (continued)

Option	Description	Default
O	Prints a message identifying any array references with out-of-bounds subscripts found during execution [†] . Enables the BOUNDS compiler directives.	OFF
P	Allows double precision. Setting OFF=P causes at compile time: 1. All double-precision declaratives to be treated as real; 2. Double-precision functions to be changed to the corresponding single-precision functions; 3. Double-precision constants to be converted as double-precision and truncated to real; 4. D's in FORMAT statement to be changed to E.	ON
Q	Aborts compilation when 100 fatal error messages counted	ON
R	Rounds results on multiply operation	ON
S	Lists FORTRAN source code	ON
T	Lists the Symbol Table after each program unit	ON
U	Enables recognition of INTEGER*2 declaration. OFF=U processes variables declared INTEGER*2 as 64-bit integers.	ON
V	Vectorizes inner DO-loops	ON
W	Compiles all floating-point operations as return jumps to user-supplied external routines. ^{††} (See table 1-3.)	OFF
X	Lists the Symbol Table with cross references after each program unit (X overrides T)	OFF
Z	Writes the Debug Symbol Table on \$BLD	OFF

[†] Bounds checking inhibits many optimizations CFT normally performs.

^{††} The W option has no effect on complex or double-precision arithmetic, intrinsic functions, or expressions in a DATA or PARAMETER statement.

OUTPUT FROM CFT

Relocatable binary output is written on the B-dataset in a format suitable for input by the system loader, LDR, one record per program unit. When requested, LDR loads and links this file plus routines required from the system or user libraries.

LISTABLE OUTPUT

CFT optionally produces a dataset containing the following.

- A source statement listing
- Error messages and their severity
- Tables of statement numbers, names encountered, external names, loops encountered, and block names and their octal lengths

The CFT control statement and compiler directives allow the user to control this output and specify the receiving dataset. Listable output is divided into pages. The number of lines per page is controlled by the LPP parameter on the OPTION control statement (see the CRAY-OS Version 1 Reference Manual, publication SR-0011).

Page header lines

Each page of listable output contains a header line with the following information.

- The name of the program unit (except for the first page for each program unit)
- The current page number within the program unit
- The truncation count, if nonzero (see the TRUNC parameter on the CFT control statement, earlier in this section)
- A list of compiler options currently turned on (see table 1-2)
- The date and time compilation began
- The CFT revision level and assembly date
- The global page number

Source statement listings

The source statement listing is generated when the S list option is selected. The listing is a record of all FORTRAN statements comprising the program as they are sequentially read and interpreted from the source input dataset. A sequence number is listed for each statement identifying its position in the program. A line number for each line is listed to the left of the sequence number. Continuation lines and comments are separate lines but not separate statements. Errors encountered during a statement compilation are flagged by lines subsequent to that statement or recorded at the end of the source statement listing.

BLOCK BEGINS messages

CFT divides program units into smaller units called blocks. These blocks are the basic units optimized by CFT. Specifying ON=B or ON=G in the CFT control statement produces a BLOCK BEGINS message for each block, listing the sequence number and relative program address of the beginning of each block. If ON=B or ON=S is selected, the message VECTOR LOOP BEGINS is listed for blocks with a vector loop.

A vector block can begin several lines before a vectorized DO-loop. (Only an innermost loop is a candidate for vectorization.) Results calculated in this loop preamble are used by the optimizer in the loop. Debugging instructions should be inserted between blocks to avoid altering the generated code of the block being tested.

Table of statement numbers

The table of statement numbers can be in a short form, excluding cross reference information (T option) or in a long form, including cross reference information (X option). In either case, the table lists all statement numbers used in the program unit, followed by a suffix indicating whether the number is inactive (SN), a FORMAT statement, or undefined (UNDEF*). For active statements, the relative address of the beginning of the statement is given.

Statement numbers are internally generated for logical IF statements, implied-DO statements, and ENTRY statements. A 5-digit number in sequence starting with 00001 is generated with leading zeros present and significant.

For DO-loops, two internally generated statement labels are created, one at the top of the loop (the reloop point) and one after loop termination control (the zero trip point). These labels are generated by suffixing the loop terminal number with letters A, B, etc., taken in pairs.

By default, internally generated numbers are not listed in this table. Specifying ON=I lists them.

Table of names encountered

This table has a short form, excluding cross reference information (T option) and a long form, including cross reference information (X option). In either case, the following fields of information are presented.

- Address
- Name
- Type
- Main usage
- Block

Address field - Values in this field are in octal and are either addresses relative to the beginning of the program, the local stack area, or a named common block; or a B or T register number.

Name field - The name field contains an alphabetized list of all symbolic names specified in the program unit. If the PROGRAM statement is omitted from the executable program, CFT identifies the main program with the name \$MAIN.

Type field - The type field gives the type of array, variable, or program unit and can contain the following.

<u>Type</u>	<u>Significance</u>
C	Complex
D	Double precision (prefix to other types, if they are double)
I	Integer (64 bits)
I'	Integer (24 bits)
L	Logical
R	Real
CH	Character
none	Typeless function or subroutine

If the item is defined or declared but not used, the type code is preceded by *.

Main usage field - An entry in this field describes the use of the corresponding symbolic name and can contain the following.

<u>Use</u>	<u>Significance</u>
n D. EQ. ARRAY	n -dimensional array in EQUIVALENCE
n DIM ARRAY	Array with n dimensions ($1 \leq n \leq 7$)
ENTRY	Entry
EQUIVALENCE	Variable or array in EQUIVALENCE
EXTERNAL	External function or subroutine
INTRINSIC	Intrinsic function
PARAMETER	Symbol appears in PARAMETER statement

<u>Use</u>	<u>Significance</u>
ST. FUNCTION	Arithmetic statement function
UNDEF EQUIV	Variable or array appears in EQUIVALENCE, but does not appear on the left side of an assignment operator in a data statement
UNDEFINED ***	Variable or array never defined
VARIABLE	Simple variable
#T-REG	Simple variable assigned to a T register instead of memory

Block field - The block field identifies the common block containing a variable or array. If no common block name appears, the variable or array is local to the program unit. If the name #ST appears, the variable or array is assigned to stack storage. If the name #T-REG or #B-REG appears, the variable is permanently assigned to a register.

If the symbol is a dummy argument to the subroutine or function, the field contains the characters DUM.ARG. and the address field contains the dummy argument number.

Table of block names and lengths in octal

This table lists the name of each block referenced in the program unit preceded by its word length in octal. The C list option controls this table.

The program block is the first block listed and it has the same name as the program unit being compiled. Pound blocks follow the program block and their names begin with a # sign. The program block and pound blocks are created by the compiler and contain code and static data to execute the compiled program unit.

The Static Space Table and Stack Space Table follow the Block Name and Length Table and describe space usage of the compiled program units. The C list option controls printing of the Static and Stack Space Tables.

The Static Space Table describes how space is used in program and pound blocks. The Static Space Table has the following entries.

B SAVE	Number of words reserved to hold values in B registers; one number greater than the number of B registers used by the generated code.
--------	---

T SAVE	Number of words reserved to hold values in T registers; equal to the number of T registers used by the generated code.
CONSTANTS	Number of words reserved to hold read-only constants.
VARIABLES	Number of words reserved for local variables, including variables declared by the user and variables created by the compiler.
TEMPORARIES	Number of words reserved to hold temporary variables. Temporary variables are compiler-generated variables and are usually reused from code block to code block.
CODE	Number of words occupied by generated code.
TOTAL	Number of static words required to execute the compiled program unit; equal to the sum of the preceding items and also the sum of the lengths of program and pound blocks.

If stack mode is requested (see the subsection on the CFT control statement for a description of the ALLOC parameter), the table describing stack space usage in the program units is printed. The Stack Space Table has the same format as the Static Space Table and contains the number of words of stack space required for the B-register save area, T-register save area, stack-based variables, and stack-based temporaries. The amount of stack space needed by the program unit is also printed.

Table of external names

This table is generated only when cross reference information (X option) is requested and contains external names and source program references.

Table of loops encountered

This table presents the following fields of information relevant for program loops when the D list option is selected.

Label	Statement number ending the loop
Index	DO-loop index
From	Beginning source line number
To	Ending source line number

Address Parcel address of loop start (blank if no loop is generated)

Length Octal number of words of code generated for the loop body.
When a loop is not generated for a DO-loop, such as a short
vector loop, the word `INLINE` appears in place of the length.

Cross reference information

Cross reference information is optionally included in the list output with the selection of the X list option. When requested, the table of statement numbers, the table of names encountered, and the table of external names include the source program references. These references are keyed to the source listing line numbers. The following codes are used in these references.

<u>Code</u>	<u>Significance</u>
A	Used in FORTRAN ASSIGN statement
D	Defined in declarative statement
E	Statement number ending a DO-loop
I	Index of a DO or implied DO-loop
J	Statement number used in transfer
L	Source line of a statement number
N	Name used as a DO-loop parameter
P	Used in CALL/FUNC call or array reference
R	Format used in READ statement
S	Stored so contents can be changed
U	Name used in executable statement
W	Format used in WRITE statement
?	Ten or more references to symbol

Messages

Up to five levels of messages are produced by CFT, depending on the E parameter on the CFT control statement. (See Appendix D for details of messages.)

Program unit page table

If more than one program unit is compiled, CFT prints a sorted table of the names of the units compiled, listing the beginning global page number of each program unit. This table appears at the end of CFT's output.

COMPILER OPTIONS

Compiler options expressed by the user in the CFT control statement (see description of the CFT control statement earlier in this section) establish particular methods for application throughout the compilation of all related FORTRAN program units. *Compiler directives* encountered in the program units being compiled can change or reinstate this set of methods. Certain other compiler actions are enabled and disabled only by compiler directives.

The CFT control statement E (enable compiler directives) option must be specified ON for compiler options to be recognized by CFT. Otherwise, the lines containing compiler options are treated as comment lines.

USING COMPILER DIRECTIVE LINES

A *compiler directive line* contains the characters CDIR\$ in columns 1 through 5. Column 6 of the initial line must be blank or contain the character 0. Columns 7 through 72 of the initial line contain zero or more compiler directives separated by commas. If the compiler directive has a list associated with it, no other compiler directive can appear on the same card. Spaces can precede, follow, or be embedded within a compiler directive. Columns 73 through 96 can be used for any purpose. Continuation of compiler directive information beyond a single line can be accomplished by one of the following methods.

- Enter any character except a blank or zero in column 6 of up to 19 subsequent lines.
- Enter the characters CDIR\$ in columns 1 through 5 of all lines in the sequence.

Comment or blank lines cannot occur within a continued CDIR\$ sequence. The first non-CDIR\$ line terminates the CDIR\$ continuation sequence.

The character C in column 1 identifies lines as comment lines to all but the Cray FORTRAN Compiler. This feature maintains the transportability of programs using compiler directives.

Compiler directive lines are listed in the source statement listing.

COMPILER DIRECTIVES

CFT provides the following categories of compiler directives.

- Listable output control
- Vectorization control
- Integer control
- Multiply/divide control
- Flow trace
- Scheduler

- Dynamic common block
- Array bounds checking
- Optimization
- Debugging

LISTABLE OUTPUT CONTROL DIRECTIVES

Following are the listable output control directives.

- EJECT
- LIST
- NOLIST
- CODE
- NOCODE

The CFT control statement options L (listable output control directives) and E (error messages) must be on to cause recognition of this set of compiler directives.

EJECT directive

A compiler directive line containing an EJECT directive is printed as the last line of the current page of source statement listing. If the EJECT directive is contained in a continuation set of compiler directive lines, the last of these becomes the last line of the page. In either case, a new page begins. The EJECT directive has no effect if production of the source statement listing has been suppressed.

The form of the EJECT directive follows.

EJECT

LIST directive

The LIST directive causes the production of a source statement listing or is ignored if one is already being produced. The LIST directive also restores the other list options specified on the CFT control statement.

If the NORECURRENCE directive is omitted, the CFT compiler executes vectorizable loops with recurrences in vector mode if the iteration count is known to be 15 or greater. Generally, vector mode is faster than scalar mode for DO-loops with recurrences. If attempted vectorization of inner DO-loops is not specified by a CFT control statement option, the NORECURRENCE directive is ignored.

IVDEP directive

The IVDEP directive is specified in advance of a DO statement to cause the compiler's attempts to vectorize the corresponding DO-loop to ignore any vector dependencies encountered. The IVDEP directive affects only the single innermost DO-loop it precedes. Note that conditions other than vector dependencies can cause the inhibiting of vectorization whether or not an IVDEP directive is specified.

The form of the IVDEP directive follows.

IVDEP

VFUNCTION directive

The VFUNCTION directive declares that a vector version of an external function exists.

The form of the VFUNCTION directive is

VFUNCTION f [, f] ...

where f is the symbolic name of a vector external function.

The function f must be written in CAL and must use the call-by-value sequence. Because CFT prefixes and suffixes the name with % as part of the calling sequence, f must be limited to six characters. (See the Macros and Opdefs Reference Manual, CRI publication SR-0012 for details on non-FORTRAN procedures.) f must not be the name of a dummy procedure.

VFUNCTION arguments must be either vectorizable expressions or scalar expressions. If the argument list contains both scalar and vector arguments in a vector loop, the scalar arguments are broadcast into the appropriate vector registers. If all arguments are scalar or the reference is not in a vector loop, the function $f\%$ is called with all arguments passed in S registers. Functions named in a VFUNCTION list

must not have side effects. (CDIR\$ VFUNCTION implies NO SIDE EFFECTS; the names of functions appearing in the VFUNCTION directive need not appear in a CDIR\$ NO SIDE EFFECTS list.) Registers are used for argument transmission and, therefore, no more than seven single-word items or three double-word items can be passed by a call. One register passes each single-word argument and two registers pass each double-word argument; these can be mixed in any order with a maximum of seven required registers.

The VFUNCTION directive must precede any statement function definitions or executable statements in a program. If the names of functions in a VFUNCTION directive also appear in an EXTERNAL declaration, the EXTERNAL declaration must precede the VFUNCTION directive.

A VFUNCTION function should receive inputs from its argument list. The VFUNCTION function should not change the value of its arguments or variables in common blocks and should not reference variables in common blocks which are also used by a program unit in the calling chain.

NEXTSCALAR directive

The NEXTSCALAR directive, specified in advance of a DO statement, causes only that DO-loop to be executed in scalar mode. Vectorization is inhibited.

The form of the NEXTSCALAR directive follows.

NEXTSCALAR

SHORTLOOP directive

The SHORTLOOP directive, specified in advance of a DO statement, states that the succeeding DO-loop will be executed at least once and at most 64 times, allowing CFT to generate special code for the succeeding DO-loop. This directive may decrease execution time because it eliminates the run time tests that determine if a vectorized DO-loop has been completed. Using this directive before a zero-iteration DO-loop or a DO-loop that should be executed more than 64 times produces indeterminate results.

The form of the SHORTLOOP directive follows.

SHORTLOOP

INTEGER CONTROL DIRECTIVES (INT24, INT64)

The specification of INT24 or INT64 in a program unit causes all variables and arrays named in its argument list to be identified as entities of type integer. When INT24 is specified, the integers provide

FLODUMP routine

FLODUMP provides, upon request, a dump of the flow trace tables when a program aborts with flow trace active. FLODUMP dumps the tables in flow trace format. FLODUMP is invoked by specifying ON=F in the CFT control statement and by including the FLODUMP control statement.

The FLODUMP control statement follows the EXIT and DUMPJOB control statements.

Example:

```
JOB, ... .  
CFT,ON=F.  
LDR.  
EXIT.  
DUMPJOB.  
FLODUMP.  
.  
.  
.
```

See the CRAY-OS Version 1 Reference Manual, publication SR-0011 for details of the FLODUMP control statement.

Options

As an additional option, the user can select one or more of the following.

- SETPLIMQ
- ARGPLIMQ
- FLOWLIM

SETPLIMQ - This option enables the flow trace routine to print a line on \$OUT for every CALL or RETURN statement executed, listing the following information.

- Routine name
- Calling routine name
- Job time
- Time the routine is entered
- Time spent in the routine
- Time the routine returns

Because this option can generate a large volume of output, it must be explicitly requested at runtime as follows.

CALL SETPLIMQ(KOUNT)

The value of KOUNT specifies the number of trace lines printed. Since one line is produced for each CALL and each RETURN, KOUNT should be set to twice the number of CALL statements for which flow trace is desired.

In effect, each CALL and each RETURN statement is given a sequence number at run time. Each subsequent CALL or RETURN statement whose sequence number is less than ABS(KOUNT) causes a printout. CALL or RETURN statements executed before the CALL SETPLIMQ(KOUNT) count toward the line limit but do not generate any output. In general, CALL SETPLIMQ(KOUNT) is one of the first executable statements in a program.

ARGPLIMQ - ARGPLIMQ enables the flow trace routine to list the subroutine arguments for the next ABS(KOUNT) calls. This option must be explicitly requested at run time as follows.

CALL ARGPLIMQ(KOUNT)

This option can be called only once in a program.

FLOWLIM - FLOWLIM enables the flow trace routine to limit the number of traced subroutines to the next ABS(KOUNT) subroutines. After this limit is reached, the flow trace summary is printed. Further calls to FLOWENTR and FLOWEXIT result in a return to the user's calling subroutine, thus reducing overhead time. In effect, the call to FLOWLIM turns off the flow trace option after the limit is reached.

The FLOWLIM option must appear before any subroutine calls and it must be explicitly requested at run time as follows.

CALL FLOWLIM(KOUNT)

KOUNT=0 traces all subroutines.

SCHEDULER DIRECTIVES

The list of scheduler directives follows.

- UNSAFEIF
- SAFEIF
- BL
- NOBL

UNSAFEIF/SAFEIF directives

The UNSAFEIF and SAFEIF directives enable or disable movement of code past the branch of an IF statement for a block of code, respectively. If UNSAFEIF is enabled, the code scheduler attempts to move any operation except a store or divide over a branch instruction. A branch instruction may have been inserted to protect the operation. UNSAFEIF allows code movement for a block of code. SAFEIF prevents code movement over an IF statement for a block of code. UNSAFEIF and SAFEIF apply to one block at a time and the last directive appearing in a block is the directive used. The CDIR\$ directives override the default or CFT control card option for one block of code.

BL/NOBL directives

The BL and NOBL directives enable or disable the prefetch of operands over a loop branch, respectively. The code scheduler usually attempts to prefetch operations in eligible short scalar loops. Subscripts for the iteration after the last one may be out of range to cause an operand range error. BL allows prefetch of code. NOBL prevents prefetch of an operand for the next block of code. The CDIR\$ directives override the default or CFT control card options. The CDIR\$ directives apply to one block at a time and the last directive appearing in a block is the directive used.

DYNAMIC COMMON BLOCK DIRECTIVE (DYNAMIC)

The DYNAMIC directive declares dynamic common blocks for users with dynamic common block capability. The COS loader does not support the dynamic common block capability.

The form of the DYNAMIC directive is

`DYNAMIC b[,b]...`

where b is the name of a previously encountered common block.

ARRAY BOUNDS CHECKING DIRECTIVE (BOUNDS)

The BOUNDS directive checks most array references for out-of-bounds subscripts. The BOUNDS directive is enabled when the ON=O parameter is specified on the CFT control statement and can be controlled by a CDIR\$ BOUNDS directive. If ON=O is not specified, all CDIR\$ BOUNDS directives are ignored.

The ON=O option is global to all program units in the compilation. The BOUNDS directives are local to the program unit where they appear.

Bounds checking typically increases program run time by a factor of 10 and inhibits vectorization of any DO-loop that references a checked array.

Bounds checking is not applied to arrays of type character or array references that appear in argument lists or in input/output statements. If an array has a last dimension of * or 1, bounds checking is not performed on the last dimension. Dependency messages issued with bounds checking turned on may not appear when bounds checking is turned off, because bounds checking is performed by passing an array argument to a nonvectorizable procedure or function. If a DO-loop contains an array being checked, a dependency message may be issued.

BOUNDS options

The BOUNDS directive can be specified with three different argument options.

- The BOUNDS directive with no arguments

BOUNDS

This option enables bounds checking for all arrays. It remains in effect until another BOUNDS directive or the end of the compilation unit is encountered.

- The BOUNDS directive with an empty argument list

BOUNDS ()

This option disables bounds checking for all arrays. It remains in effect until another BOUNDS directive or the end of the compilation unit is encountered.

- The BOUNDS directive with an argument list

BOUNDS (*a, b, c*)

This option enables bounds checking only for the arrays named in the argument list and remains in effect only for the current routine, or until another BOUNDS directive is encountered. Bounds checking can be enabled and disabled many times in a specific compilation unit. Bounds checking for all arrays is performed in subsequent program units until another BOUNDS directive is encountered.

OPTIMIZATION DIRECTIVES

The following directives are optimization directives.

- BLOCK
- NO SIDE EFFECTS
- ALIGN
- NOIFCON
- RESUMEIFCON
- RESUMEDOREP
- NODOREP

BLOCK directive

The CFT compiler divides source code into sections called blocks. The BLOCK directive, specified in advance of a FORTRAN statement, causes a block to begin with the succeeding FORTRAN statement.

Blocks are used as the basis for optimization and vectorization by the compiler. This directive is useful for machine-timing tests and for certain unusual program debugging applications.

NO SIDE EFFECTS directive

The NO SIDE EFFECTS directive declares that an external subprogram has no side effects. A NO SIDE EFFECTS external subprogram does not redefine the value of a variable local to the calling program, passed as an argument to the subprogram, or declared in a common block. Using the NO SIDE EFFECTS directive allows CFT to keep information in registers across subprogram invocations without reloading the information from memory after returning from the subprogram. Intrinsic functions are assumed to have no side effects.

The form of the NO SIDE EFFECTS directive is

NO SIDE EFFECTS f [, f]...

where f is the symbolic name of an external subprogram the user guarantees to have no side effects. f must not be the name of a dummy procedure.

A NO SIDE EFFECTS subprogram should receive inputs from its arguments. The subprogram should not reference or define variables in a common block shared by a program unit in the calling chain, or redefine the value of its arguments. If these conditions are not met, results can be unpredictable.

The NO SIDE EFFECTS directive must precede arithmetic statement functions or executable statements in a program. If the name of a subprogram appears in a NO SIDE EFFECTS directive and an EXTERNAL declaration, the EXTERNAL declaration must precede the NO SIDE EFFECTS directive.

CFT may move invocations of a NO SIDE EFFECTS subprogram from the body of a DO-loop to the loop introduction if the arguments to that function are invariant in the loop. This may affect the results of the program, particularly if the NO SIDE EFFECTS subprogram calls functions like the random number generator or the real-time clock.

ALIGN directive

The ALIGN directive causes the next referenced statement label, the first instruction of the next DO-loop body, or the next ENTRY point to align on an instruction buffer boundary. The beginning of a DO-loop, a referenced statement label, or an ENTRY point will be aligned. The ALIGN directive must appear immediately before the aligned statement.

If the ALIGN directive does not immediately precede a SUBROUTINE statement, PROGRAM statement, FUNCTION statement, ENTRY statement, DO statement, or a statement with a referenced statement label, a warning message is issued and the directive is ignored.

CFT does not generate a loop construct for short vector loops; therefore, these loops are not aligned. If an ALIGN directive appears before a short vector loop, a warning message is issued and the directive is ignored.

The ALIGN directive is useful for fitting loops and short subprograms into instruction buffers, so the buffer will not need frequent reloading.

The form of the ALIGN directive follows.

ALIGN

NOIFCON directive

The NOIFCON directive disables optimization of conditional replacement statements of the form *IF(logical exp) var=expression*, except when the statement can be converted to a MAX/MIN function. Conditional replacement statements appearing before a NOIFCON directive can be optimized at the level specified on the CFT control card. Optimization is disabled only for conditional replacement statements appearing after the NOIFCON directive. The NOIFCON directive is ignored if the optimization level is NOIFCON.

The form of the NOIFCON directive follows.

NOIFCON

RESUMEIFCON directive

The RESUMEIFCON directive enables optimization of conditional replacement statements at the level specified by the OPT=*option* on the CFT control card. When this optimization is enabled, CFT attempts to optimize statements of the form IF(*logical exp*) var=*expression* by producing code similar to that for var=CVMGX(*expression*, var, *condition*). If NOIFCON is specified on the control card either by default or with an OPT=NOIFCON parameter or if the optimization has not been disabled by a CDIR\$ NOIFCON directive, the RESUMEIFCON directive is ignored.

The form of the RESUMEIFCON directive follows.

RESUMEIFCON

RESUMEDOREP directive

The RESUMEDOREP directive, specified before a DO statement, enables replacement of successive 1-line DO-loops by calling a \$SCILIB routine at the level specified by the OPT parameter on the CFT control statement. If OPT=NODOREP, CDIR\$ RESUMEDOREP is ignored. Part 3, section 2 has examples on the use of this directive. OPT=SAFEDOREP is the default option.

The form of the RESUMEDOREP directive follows.

RESUMEDOREP

NODOREP directive

The NODOREP directive disables replacement of 1-line DO-loops with a call to a \$SCILIB routine until a RESUMEDOREP directive is used. Specifying NODOREP has no effect on vectorization of successive DO-loops.

The form of the NODOREP directive follows.

NODOREP

DEBUGGING DIRECTIVES (DEBUG, NODEBUG)

DEBUG and NODEBUG are debugging directives enabling or disabling the generation of sequence number labels, respectively. The DEBUG and NODEBUG directives are recognized only when the DEBUG parameter is specified on the CFT control statement. See the Symbolic Interactive Debugger User's Guide, CRI publication SG-0056 for a detailed description on the debugging directives.

DEBUG writes sequence number labels for executable FORTRAN statements to the Debug Symbol Table, allowing breakpoints to be set with the Symbolic Interactive Debugger (SID) at statement sequence numbers. DEBUG disables vectorization and scheduling.

NODEBUG disables sequence number label generation and restores vectorization and scheduling.

EXTERNAL ROUTINES

The external routines shown in table 1-3 are called with the call-by-value sequence. Normally the first operand is in S1 or V1, the second is in S2 or V2, and results are returned in S1 or V1. An exception is the divide routines, where the reciprocal approximation to S2 or V2 is returned in S1 or V1.

Table 1-3. External routines

Operation	External routine
s + s	RASS%
s - s	RSSS%
s * s	RMSS%
l / s	RDSS%
s + v	RASV%
s - v	RSSV%
s * v	RMSV%
l / v	RDSV%
v + v	RAVV%
v - v	RSVV%
v * v	RMVV%
v - s	RSVS%

v = vector

s = scalar

NOTE

The following operations are changed as indicated
before the external routine is called.

$v + s$	$s + v$	s / v	$(1/v) * s$
$v * s$	$s * v$	$-s$	$0 - s$
v / s	$(1/s) * v$	$-v$	$0 - v$
$v1 / v2$	$(1/v2) * v1$		

- Vector array reference - An array element where one subscript expression contains one CII reference and where any other subscript expression is an invariant expression. The subscript expression containing the CII must be a linear expression algebraically reducible to the following form

$$[+INvariant\ Expression_1*]CII[+INvariant\ Expression_2]$$

where the only operators, if any, in invariant expression₁ are multiply operators. Use of parentheses in the CII subscript expression will prevent vectorization. The subscript portion of a subscript reference is a vectorizable expression and partially vectorizes; that is, CFT generates a separate scalar loop to handle the subscript reference and vectorize the remaining loop.

- Scalar temporary - A variable set equal to a vectorizable expression during each pass through a DO-loop
- Variable or invariant array element used in a reduction array operation. The item must appear on both the left and the right sides of the equal sign. On the right, it must be a summand, multiplicand, dividend, or minuend. For example,

```
X = X+A(I)*B
X = X-A(I)*B

Y = Y*(A(I)+4)
Y = Y/(A(I)+4) .
```

The type of reduction variable must be INTEGER or REAL. Real operations between the reduction variable and the remaining expression are limited to addition, subtraction, multiplication, and division. Integer operations are limited to addition and subtraction. No other operations are allowed between the reduction variable and the remaining expression.

- Pseudo vector - An array reference without the previous requirements, a subscript expression that vectorizes, and no dependencies will pseudo vectorize. The subscript expression is computed as a vector expression and the reference is treated as a scalar subloop inside the vector loop.

In the following example, I, J, and K are CIIs; A, B, and C are vector array references; KDELTA, 107, 3, 2, 7, M, L, and X are invariants; and D(L,M) is an invariant array element.

```
DO 10 I = 3,101,2
  K = K - KDELTA
  J = 107 - I
  A(3,I-2) = COS(B(J)) **C(M-2*K+L*M/7,L,M/L)*X*D(L,M)
10  CONTINUE
```

DEPENDENCIES

CFT inhibits vectorization of DO-loops with dependencies. The following example of a DO-loop shows a dependency within CFT. In this example, the first seven elements of array A are 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, and 7.0, respectively.

```
      DO 10 I = 2,7  
      A(I) = A(I-1)  
10    CONTINUE
```

The results of array A differ, depending on the mode type the loop is executing, vector mode or scalar mode. Table 2-1 shows the first seven elements of array A in vector and scalar modes.

Table 2-1. Array A elements in vector and scalar modes

Array A elements	1	2	3	4	5	6	7
Vector mode	1.0	1.0	2.0	3.0	4.0	5.0	6.0
Scalar mode	1.0	1.0	1.0	1.0	1.0	1.0	1.0

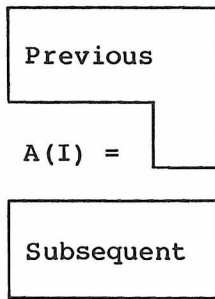
The scalar results are correct by default. CFT detects that the vector results may be different from the scalar results for the DO-loop and inhibits vectorization. A dependency exists if the following two conditions are met.

- An array is referenced and defined in the DO-loop
- An array element defined in a previous pass of the DO-loop is referenced

CFT may internally change the order of the definition and reference, eliminating an apparent dependency.

In the previous example, A(I) is the definition and A(I-1) is the reference. A definition is on the left side of the = operator. The dependency detected is called a previous minus with an incrementing subscript.

The term previous means a reference in a statement occurs before the definition statement, or a reference is on the right side of the = operator. The term subsequent means a reference in a statement occurs after the definition statement. The following diagram describes the terms previous and subsequent.



The following example also has a previous reference because the reference, $A(I-1)$, is in a statement before the definition $A(I)$.

```
DO 20 I = 2,M
  B(I) = A(I-1)
  A(I) = C(I)
20  CONTINUE
```

The term minus means the subscript of the reference ($I-1$) is less than the subscript of the definition (I). In this example, subscript I is incrementing. Both of the examples have a previous minus with an incrementing subscript dependency.

The following information is required to determine if there is a dependency.

- Previous or subsequent reference
- Plus or minus subscript difference
- Incrementing or decrementing subscript

Table 2-2 shows dependency information combinations and results.

Table 2-2. Dependency information combinations

Dependency information combinations	Results
Previous minus with an incrementing subscript	Dependency
Previous minus with a decrementing subscript	Vector
Previous plus with an incrementing subscript	Vector
Previous plus with a decrementing subscript	Dependency
Subsequent minus with an incrementing subscript	Vector
Subsequent minus with a decrementing subscript	Vector
Subsequent plus with an incrementing subscript	Vector
Subsequent plus with a decrementing subscript	Vector

The following two DO-loops have previous plus with decrementing subscript dependencies.

```
      DO 30 I = 99,1,-1
      A(I) = A(I+1)
30    CONTINUE
```

```
      DO 40 I = 99,N,-1
      B(I) = A(I+1)
      A(I) = C(I)
40    CONTINUE
```

The following two DO-loops also have dependencies.

```
      DO 50 I = 1,99
      A(I) = B(I)
      A(I+1) = C(I)
50    CONTINUE
```

```
      DO 60 I = 99,1,-1
      A(I) = B(I)
      A(I-1) = C(I)
60    CONTINUE
```

In these two examples, A never appears on the right side of the = operator, therefore, assigning the reference and definition labels is ambiguous. In this case, it is assumed the last statement has the definition. The definitions for the first and second examples are A(I+1) and A(I-1), respectively. This means the first example has a previous minus with an incrementing subscript dependency and the second example has a previous plus with a decrementing subscript dependency.

This example is a previous plus with a decrementing subscript dependency.

```
      DO 10 I = 1,10
      A(11-I) = A(12-I)
10    CONTINUE
```

The DO control variable is an incrementing I and the subscripts (11-I) and (12-I) are decrementing. Therefore, this DO-loop would cause CFT to detect a previous plus with a decrementing subscript dependency.

The following DO-loops do not have dependencies.

Previous minus with a decrementing subscript

```
      DO 10 I = 100,2,-1
      A(I) = A(I-1)
10    CONTINUE
```

Previous plus with an incrementing subscript

```
DO 20 I = 1,99
  B(I) = A(I+1)
  A(I) = 3.0
20  CONTINUE
```

Subsequent minus with an incrementing subscript

```
DO 30 I = 2,100
  A(I) = B(I)
  C(I) = A(I-1)
30  CONTINUE
```

Subsequent minus with a decrementing subscript

```
DO 40 I = 100,2,-1
  A(I+1) = B(I)
  C(I) = A(I)
40  CONTINUE
```

Subsequent plus with an incrementing subscript

```
DO 50 I = 1,100
  A(I) = B(I)
  C(I) = A(I+1)
50  CONTINUE
```

Subsequent plus with a decrementing subscript

```
DO 60 I = 2,100
  A(I-1) = B(I)
  C(I) = A(I)
60  CONTINUE
```

Two types of dependency messages are issued by CFT.

```
AT SEQUENCE NUMBER - m
PRNAME name COMMENT - DEPENDENCY INVOLVING ARRAY "name"
```

is issued when the definition and reference appear in the same statement with sequence number *m*.

```
AT SEQUENCE NUMBER - m
PRNAME name COMMENT - DEPENDENCY INVOLVING ARRAY "name" IN
SEQUENCE NUMBER n
```

is issued when the definition appears in a statement with sequence number *m* and the reference appears in a statement with sequence number *n*.

When a dependency message is issued, an informative dependency message also appears, explaining why the dependency exists. See Appendix D for a description of these messages.

All the previous examples had a subscript incremented or decremented by 1. The following three examples have DO-loops with an increment and decrement other than 1.

```
      DO 10 I = 10,20,2
      A(I) = A(I-1)
10    CONTINUE
```

The first example has a previous minus with an incrementing CII dependency. Because the increment is 2, A(I) and A(I-1) never access the same array elements. A(I) accesses elements 10, 12, 14, 16, 18, and 20. A(I-1) accesses elements 9, 11, 13, 15, 17, and 19. This type of DO-loop vectorizes.

```
      DO 20 I = 20,10,-2
      A(I) = A(I+3)
20    CONTINUE
```

This example also has a dependency but A(I) and A(I+3) access different array elements. This DO-loop also vectorizes.

```
      DO 30 I = 10,20,2
      A(I) = A(I-4)
30    CONTINUE
```

In this example, A(I) and A(I-4) access some of the same array elements. A(I) accesses elements 10, 12, 14, 16, 18, and 20. A(I-4) accesses 6, 8, 10, 12, 14, and 16. In this case, a dependency message is issued.

CFT cannot always determine if the subscript is incrementing or decrementing as shown in the following example.

```
      DO 40 I = M,N,J
      A(I) = A(I+2)
40    CONTINUE
```

If J is positive, the subscript is incrementing and there is no dependency. If J is negative, the subscript is decrementing and a previous plus with a decrementing subscript dependency exists. This increment is ambiguous and causes a dependency message to be issued.

CFT cannot always determine if the subscript difference is minus or plus.

```
      DO 50 I = 1,100
      A(I) = B(I)
      C(I) = A(I+J)
50    CONTINUE
```

If J is positive, the subscript difference is plus. If J is negative, the subscript difference is minus. This example also causes a dependency message to be issued.

If the value of J is known not to cause a dependency, an IVDEP compiler directive can be used, forcing CFT to generate vector code for a DO-loop with dependencies (see part 3, section 1 for the format of compiler directives).

Equivalenced arrays can introduce different dependency problems related to the storage overlap, as in the following example.

```
      DIMENSION A(6),B(6),X(6)
      EQUIVALENCE (B,A(3))
      DO 5 I = 2,5
      A(I) = ...
      X(I) = B(I)
5     CONTINUE
```

This sequence causes the multiple-statement dependency message to be printed. The message refers to the dependency between A(I) and B(I). The message uses only one array name rather than both names. In general, messages concerned with equivalenced arrays print only the first name encountered in processing the declarative statement.

The inhibiting of vectorization because of such dependencies can be relaxed in the case of multiply-dimensioned array processing. CFT must be able to determine that the specified array elements are in different vectors (that is, rows, columns, planes, etc.) of the array. For example, the loop

```
      DO 10 I = 2,100
10     A(I,J) = A(I-1,J-1)
```

is vectorizable, while the similar loop

```
      DO 20 I = 2,100
20     A(I,J) = A(I-1,JMINUS1)
```

is not vectorized since CFT cannot determine whether J and JMINUS1 are equal.

The compiler directive IVDEP can be placed in advance of an inner DO-loop DO statement to cause vector dependencies to be ignored in determining whether or not to vectorize that loop. (See part 3, section 1 for a description of IVDEP and other compiler directives.)

VECTORIZATION WITH ARRAYS

Because CFT allows only one subscript in an array reference to be variant, loops that reference the diagonal of an array are not fully vectorized. A loop such as

```
      DIMENSION A(N,N)
      DO 10 I = 1,N
10    A(I,I) = ...
```

can be rewritten as

```
      DIMENSION A(N,N)
      J = 1
      DO 10 I = 1,N
      A(J)= ...
10    J = J+N+1
```

or as

```
      DIMENSION A(N,N),B(N*N)
      EQUIVALENCE (A,B)
      J = -N
      DO 10 I = 1,N
      J = J+N+1
10    B(J) = ...
```

The first case is allowed by CFT but the use of one subscript rather than two causes a warning level diagnostic. Since array operations typically execute in times proportional to N^2 and diagonal operations execute in times proportional to N , vectorizing the diagonal operations might not have a significant effect on overall program execution time.

CFT allows variables and array elements to be defined within a vectorized loop as CIIs, scalar temporaries, or as recursively defined terms. (CIIs are discussed earlier in this section.) A scalar temporary is a variable set equal to a vectorizable expression. Recursively defined terms must be defined by integer addition or subtraction or real addition, subtraction or multiplication.

The following loop is vectorizable.

```
      REAL A(100),B(100,100),C(100)
      INTEGER II(100)
      DO 10 I = 1,100
      T = B(7,I) + A(I)*B(I,7)
      C(7) = C(7) + T*SQRT(T)
      T = B(I,7)*B(11,I)
      PROD = PROD*(T + A(101-I))
      II(I) = II(I) + 1
10    ISUM = ISUM + II(I)
```


The following example describes pseudo vectorization. I, J, and K are CIIs and A, B, C, II, and JJ are arrays without dependencies.

```
A(II(I)) = B(JJ(I))  
K = C(I/J)  
Y = A(INT(SIN(B(I))*X))
```

USING OPTIMIZED ROUTINES

The efficiency of the vectorization depends on the number of iterations of the loop and the complexity of the loop. In many cases, a loop with a large number of iterations and simple calculations producing a single scalar result (for example, a dot product or a sum) should be replaced with a call to an optimized routine in the \$SCILIB library. The vector sum

```
SUM = 0.0  
DO 10 I = 1,100  
SUM = SUM + A(I)
```

is better written as

```
SUM = SSUM(100,A(1),1).
```

Nested DO-loops producing a vector result should also be replaced with a call to an optimized routine in the \$SCILIB library. For example, the following matrix multiply

```
DO 10 I = 1,N  
DO 10 J = 1,M  
A(I,J) = 0.0  
DO 10 L = 1,K  
A(I,J) = A(I,J) + B(I,L) * C(L,J)
```

is better written as

```
CALL MXM(B,N,C,K,A,M).
```

For more examples on optimized routines in \$SCILIB library see the Library Reference Manual, CRI publication SR-0014.

USE OF OPTIMIZED ROUTINES BY CFT

Several 1-line DO-loops are recognized by CFT and automatically replaced by a call to the proper library routine. Vector sums and vector dot products are replaced by calls to library routines SSUM and SDOT, respectively. The vector sum

```
      DO 10 I = 1,N
10    S = S + A(I)
```

is automatically replaced by

```
      S = S + SSUM(N,A(1),1)
```

and the vector dot product

```
      DO 10 I = 1,N
10    C(J,K) = C(J,K) + A(I,K) * B(I,K)
```

is automatically replaced by

```
      C(J,K) = C(J,K) + SDOT(N,A(1,K),1,B(1,K),1).
```

For more information on SSUM and SDOT, see the Library Reference Manual, CRI publication SR-0014.

1-line DO-loops calculating a single vector result (for example, first order linear recurrences) are recognized and automatically replaced by a call to the \$SCILIB library FOLR, FOLR2, FOLRP, and FOLR2P. The example,

```
      DO 10 I = 2,N
10    B(I) = B(I) - A(I) * B(I-1)
```

is automatically replaced by

```
      CALL FOLR(N,A(1),1,B(1),1).
```

Similarly

```
      DO 10 I = 3,500,2
10    C(I) = B(I) - A(I) * C(I-2)
```

is automatically replaced by

```
      CALL FOLR2(500,A(1),2,B(1),2,C(1),2).
```

Routines FOLRP and FOLR2P are called when the DO-loop statement's additive operation is addition instead of subtraction. For more information on first-order linear recurrences, see the Library Reference Manual, CRI publication SR-0014.

Follow these guidelines for writing 1-line DO-loops that will be optimized by replacement with library calls.

- The DO-loop body must be one and only one FORTRAN statement long. The terminating statement number can be on the same line or on the following line with a CONTINUE statement.
- The 1-line DO-loop body must be a vector sum, vector dot product, or a first order linear recurrence.
- All terms must be single-precision real and not equivalenced.
- Keep array subscripts simple; that is, of the form $A(\text{invariant} * \text{variable} + \text{invariant})$. Other loops that vectorize are less restrictive with subscript complexity than 1-line DO-loop replacement.
- The DO-loop increment (m_3) must be a positive constant value.

Conditional statements

IF statements usually break optimization blocks and, therefore, inhibit the vectorization of DO-loops. The intrinsic functions CVMGT, CVMGP, CVMGM, CVMGZ, CVMGN, MAX and MIN can often be used in lieu of IF statements to prevent breaking an optimization block.

CFT can perform some of the optimizations. IF statements of the form $\text{IF}(\text{var.op.expression}) \text{var}=\text{expression}$ can be optimized to produce code similar to $\text{var}=\text{function}(\text{var},\text{expression})$ where *function* is a version of the MAX/MIN functions. For this form of optimization to occur, *op.* must be one of the relational operators .GT., .GE., .LT., or .LE.; *var* and *expression* must be the same type, either REAL, INTEGER, or DOUBLE PRECISION. Examples of IF statements optimized by CFT in this way are

```
IF(A(I).GT.B(I))A(I)=B(I)
IF(I1.LE.I2)I2=I1
IF((I+3)*R1.GT.R2)R2=(I+3)*R1.
```

A more general form of conditional replacement statements can also be optimized by CFT. Statements of the form $\text{IF}(\text{logical exp}) \text{var}=\text{expression}$ can be optimized to produce code similar to that for $\text{var}=\text{CVMGx}(\text{expression},\text{var},\text{condition})$ where CVMGx is a vector merge function, *var* is type INTEGER, REAL, or LOGICAL, and *condition* is a logical expression. Examples of IF statements which CFT can optimize in this manner are

```
IF(COND1.OR.COND2)B(I)=C(I)
IF((B(I).GT.C(I)).OR.(B(I).LT.A(I)))B(I)=ABS(A(I)*C(I))
IF(I.GT.R)I=R (the types are mismatched so a MAX/MIN optimization
will not occur).
```

Conditional replacement statements of the form

IF(*cond*) *var*=*var op exp*

where *op* is the operator +, -, *, or /; and

IF(*cond*) *var*=*exp op var*

where *op* is the operator + or * may be restructured as if written as

var=*var op*CVMGX(*exp, ident, cond*)

where *ident* is 0 when *op* is + or - and *ident* is 1 when *op* is * or /. This form of restructuring occurs only if *exp* does not contain unparenthesized operators of lower priority than *op*. If *op* is - or /, *exp* cannot contain unparenthesized operators of the same priority as *op*. This form of the IF conversion allows a vector reduction loop to be generated if *var* is a scalar reference and the statement appears in a loop that otherwise would be vectorizable.

There are two possible drawbacks to performing this type of optimization in all cases. An illegal operation may occur. For example, IF(X.NE.0.0)R=R/X. If optimization occurred, an error would occur if X were equal to zero. Another example is, IF(X.GE.0.0)R=SQRT(X). An error would occur if X is negative.

The second drawback occurs when the IF statement appears in a DO-loop and the logical expression is usually false. For example,

```
DO 10, I = 1,100
  A(I) = B(I)*C(I)
  IF(A(I).GT.RMAX)A(I) = MAX(B(I),C(I))
30  CONTINUE
```

In this example, if A(I) were less than RMAX and the IF statement was not optimized, MAX(B(I),C(I)) would never be evaluated. If the IF statement were optimized, MAX(B(I),C(I)) would always be evaluated.

Because of these drawbacks, the user can control vector merge optimizations. By specifying OPT=NOIFCON on the CFT control card, this form of optimization is disabled (the default level of optimization). OPT=PARTIALIFCON allows the optimization to occur when the replacement expression does not involve division or an external function reference. OPT=FULLIFCON enables the optimization for all cases, including those involving division or external functions. The compiler directives CDIR\$ NOIFCON and CDIR\$ RESUMEIFCON may be used to turn the optimization off and on around unsafe cases (such as, division by zero if OPT=FULLIFCON) or when the logical expression of the IF statement is usually false. The following examples may be helpful in performing optimizations.

The simple case

```
      DO 10 I = N,M
      X(I) = C(I)
      IF(B(I).GT.C(I))X(I) = B(I)
10    CONTINUE
```

could be rewritten as

```
      DO 10 I = N,M
      X(I) = CVMGT(B(I),C(I),B(I).GT.C(I))
10    CONTINUE
```

or as

```
      DO 10 I = N,M
      X(I) = AMAX1(B(I),C(I))
10    CONTINUE
```

to produce vectorizable loops.

Similarly,

```
      DO 10 I = N,M
      IF(X(I).GE.10.)X(I) = X(I) + 1.0
10    CONTINUE
```

could be rewritten as

```
      DO 10 I = N,M
      X(I) = CVMGP(X(I) + 1.0,X(I),X(I)-10.)
10    CONTINUE.
```

The subroutines SENSEFI, SETFI, and CLEARFI can be used to control floating-point interrupts on a loop-by-loop basis and MAX or MIN functions can be used to protect function references.

Example:

```
      .
      .
      .
      CALL SENSEFI (MODE)
      CALL CLEARFI
      DO 10 I = 1,100
      X(I) = CVMGN(1./X(I),X(I),X(I))
      Y(I) = CVMGP(SQRT(AMAX1(Y(I),0.0)),Y(I),Y(I))
10    CONTINUE
      IF(MODE.NE.0) CALL SETFI
```

•
•
•

GENERAL GUIDELINES FOR VECTORIZATION

Follow the general guidelines given below to promote vectorization of DO-loop operations.

- Keep subscripts simple and explicit; do not use parentheses in subscripts.
- Do not use GO TO or CALL statements.
- Do not use IF statements which are not conditional replacement statements, such as IF(*logical exp*) *var=expression*. Many conditional replacement statements do not inhibit vectorization. Selecting OPT=PARTIALIFCON or FULLIFCON on the CFT control card enables this optimization. For more information, see the subsection on using optimized routines described earlier in this section.
- Use the Cray FORTRAN intrinsic functions where appropriate.
- Make judicious use of the Cray FORTRAN intrinsic functions CVMGT, CVMGP, CVMGM, CVMGZ, CVMGN, and the MAX and MIN functions instead of IF statements. For more information, see the subsection on using optimized routines described earlier in this section.
- Rewrite large loops containing a few unvectorizable statements as two or more loops, one or more of which will vectorize.

The Cray FORTRAN intrinsic functions described in this appendix have been grouped according to general purpose (that is, trigonometric, exponential, etc.). This grouping is for convenience and is not provided for in the ANSI FORTRAN Standard. These intrinsic functions include all Basic External Functions as described separately in ANSI FORTRAN and Cray FORTRAN extensions.

Entity types in this table are abbreviated: integer (I), real (R), double precision (D), complex (C), logical (L), Boolean (B),[†] character (CH), and Hollerith (H). Unless noted, 24-bit integer variables can be used as arguments to a function accepting integer arguments. 24-bit variables are sign extended and treated as 64-bit variables.

Notations *Full*, *Pseudo*, and *None* indicate the vector status of each subprogram. *Full* indicates the routine uses vector hardware. *Pseudo* indicates the routine uses only scalar hardware but simulates vectorization to gain efficiency for the DO-loop where such a routine appears. *None* indicates only a scalar version exists.

Table B-1 lists generic names and their corresponding specific names. See the Library Reference Manual, CRI publication SR-0014, for additional information on the use of the external routines.

[†] Boolean functions must be used with caution. If two Boolean results are combined, they are treated as integers.

Function	Definition	ANSI/ CRI extension	Function (y)		Argument(s) (x)			Vector	Code Generated
			Name	Type	No	Type	Range		

GENERAL ARITHMETIC FUNCTIONS

Truncation	$y = [x]$ Function lost; no rounding	ANSI	AINT DINT	R D	1 1	R D	$ x < 2^{46}$ $ x < 2^{95}$	Full Pseudo	Inline External
Nearest whole number	$y = [x+.5]$ if $x \geq 0$ $y = [x-.5]$ if $x < 0$	ANSI	ANINT DNINT	R D	1 1	R D	$ x < 2^{46}$ $ x < 2^{95}$	Full Pseudo	Inline External
Nearest integer	$y = [x+.5]$ if $x \geq 0$ $y = [x-.5]$ if $x < 0$	ANSI	NINT DNINT	I I	1 1	R D	$ x < 2^{46}$	Full Pseudo	Inline External
Absolute value	$y = x $	ANSI	IABS ABS DABS CABS [†]	I R D C	1	I R D C	$ x < \infty$ $ x_r , x_i < \infty$	Full	Inline External
Divide for remainder of x_1/x_2	$y = x_1 - x_2[x_1/x_2]$	ANSI	MOD AMOD DMOD	I R D	2 2 2	I R D	$ x_1 < 2^{63}$ $0 < x_2 < 2^{63}, 2^{-63} < x_1/x_2 < 2^{63}$ $ x_1 < 2^{47}$ $0 < x_2 < 2^{47}, 2^{-47} < x_1/x_2 < 2^{47}$ $ x_1 < 2^{95}$ $0 < x_2 < 2^{95}, 2^{-95} < x_1/x_2 < 2^{95}$	Pseudo Full Pseudo	External Inline External
Transfer sign	$y = x_1 $ if $x_2 \geq 0$ or $y = - x_1 $ if $x_2 < 0$	ANSI	ISIGN SIGN DSIGN	I R D	2	I R D	$ x_1 , x_2 < \infty$	Full	Inline
Positive difference	$y = x_1 - x_2$ if $x_1 \geq x_2$ $y = 0$ if $x_1 < x_2$	ANSI	IDIM DIM DDIM	I R D	2	I R D	$ x_1 , x_2 < \infty$	Full Pseudo	Inline External
Double-precision product	$y = x_1 * x_2$	ANSI	DPROD	D	2	R	$ x_1 , x_2 < \infty$	Full	Inline
Imaginary portion of complex value	$y = x_i$	ANSI	AIMAG [†]	R	1	C	$ x_r , x_i < \infty$	Full	Inline
Conjugate of complex value	$y = x_r - i * x_i$	ANSI	CONJG [†]	C	1	C	$ x_r , x_i < \infty$	Full	Inline
Obtain random number	$y =$ The first or next in a series of random numbers ($0 \leq y < 1$)	CRI	RANF	R	0			Full	External
Obtain random seed	The currently used random number seed	CRI	RANGET	I	1	I	$ x < \infty$	None	External
Establish random number seed	$y = x$	CRI	RANSET	R	1	B	$ x < \infty$	None	External
Input/output	Returns starting address of current record in dataset associated with unit x	CRI	GETPOS	B	1	I ^{††}		None	External

[†] $x = x_r + ix_i$

^{††} Can be a unit number or a Hollerith unit name

Function	Definition	ANSI/ CRI extension	Function (y)		Argument(s) (x)			Vector	Code Generated
			Name	Type	No	Type	Range		

EXPONENTIAL FUNCTIONS

Square root	$y=x^{\frac{1}{2}}$	ANSI	SQRT DSQRT CSQRT [†]	R D C	1	R D C	$0 \leq x < \infty$ $x_r \geq 0, x_i < \infty$	Full Pseudo	External
Exponent	$y=e^x$	ANSI	EXP DEXP CEXP [†]	R D C	1	R D C	$ x < 2^{13} \cdot \ln 2$ $\{ x_r < 2^{13} \cdot \ln 2\}$ $\{ x_i < 2^{24}\}$	Full Pseudo	External

LOGARITHMIC FUNCTIONS

Natural logarithm	$y=\ln(x)$	ANSI	ALOG DLOG CLOG [†]	R D C	1	R D C	$0 < x < \infty$	Full Pseudo	External
Common logarithm	$y=\log(x)$	ANSI	ALOG10 DLOG10	R D	1	R D	$0 < x < \infty$	Full Pseudo	External

TRIGONOMETRIC FUNCTIONS (Angles are in radians)

Sine	$y=\sin(x)$	ANSI	SIN DSIN CSIN [†]	R D C	1	R D C	$ x < 2^{24}$ $ x < 2^{48}$ $\{ x_r < 2^{24}, \ln 2\}$ $\{ x_i < 2^{13} \cdot \ln 2\}$	Full	External
Cosine	$y=\cos(x)$		COS DCOS CCOS [†]	R D C	1	R D C	$ x < 2^{24}$ $ x < 2^{48}$ $\{ x_r < 2^{24}, \ln 2\}$ $\{ x_i < 2^{13} \cdot \ln 2\}$	Full	External
Tangent	$y=\tan(x)$	ANSI	TAN DTAN	R D	1	R D	$ x < 2^{24}$	Pseudo	External
Cotangent	$y=\cot(x)$	CRI	COT DCOT	R D	1 1	R D	$ x < 2^{24}$	Pseudo	External
Arcsine	$y=\arcsin(x)$	ANSI	ASIN DASIN	R D	1 1	R D	$ x \leq 1$	Pseudo	External
Arccosine	$y=\arccos(x)$	ANSI	ACOS DACOS	R D	1 1	R D	$ x \leq 1$	Pseudo	External
Arctangent	$y=\arctan(x)$ $y=\arctan(x_1/x_2)$	ANSI	ATAN DATAN ATAN2 DATAN2	R D R D	1 1 2 2	R D R D	$ x < \infty$ $ x_1 < \infty$ $ x_2 \neq 0$	Pseudo	External
Hyperbolic sine	$y=\sinh(x)$	ANSI	SINH DSINH	R D	1	R D	$ x < 2^{13} \cdot \ln 2$	Pseudo	External
Hyperbolic cosine	$y=\cosh(x)$	ANSI	COSH DCOSH	R D	1 1	R D	$ x < 2^{13} \cdot \ln 2$	Pseudo	External
Hyperbolic tangent	$y=\tanh(x)$	ANSI	TANH DTANH	R D	1 1	R D	$ x < 2^{13} \cdot \ln 2$	Pseudo	External

[†] $x=x_r+i \cdot x_i$

Function	Definition	ANSI/ CRI extension	Function (y)		Argument(s) (x)			Vector	Code Generated
			Name	Type	No	Type	Range		

MAXIMUM/MINIMUM FUNCTIONS

Select maximum value	$y = \text{The largest of all } x$	ANSI	MAX0 AMAX1 DMAX1 AMAX0 MAX1	I R D R I	64	n n n n n	I R D I R	$ x < \infty$	Full	Inline
Select minimum value	$y = \text{The smallest of all } x$	ANSI	MIN0 AMIN1 DMIN1 AMIN0 MIN1	I R D R I	64	n n n n n	I R D I R	$ x < \infty$	Full	Inline

CHARACTER FUNCTIONS

Lexically greater than or equal	$y = a_1 \geq a_2$	ANSI	LGE	L	2	CH	Any legal character string	None	External
Lexically greater than	$y = a_1 > a_2$	ANSI	LGT	L	2	CH	Any legal character string	None	External
Lexically less than or equal	$y = a_1 \leq a_2$	ANSI	LLE	L	2	CH	Any legal character string	None	External
Lexically less than	$y = a_1 < a_2$	ANSI	LLT	L	2	CH	Any legal character string	None	External
Length of character entity		ANSI	LEN	I	1	CH	Any legal character string	None	Inline
Index of a substring		ANSI	INDEX	I	2	CH	Any legal character string	None	External

TYPE CONVERSION

Conversion to integer	truncation toward zero (fraction, lost)	ANSI	INT	I I I I I IFIX IDINT	1 1 1 1 1 1 1	C I* I R B** R, B** D	$ x_r < 2^{46}$ $ x < 2^{23}$ $ x < 2^{63}$ $ x < 2^{46}$ $ x < 2^{63}$ $ x < 2^{46}$	Full	Inline
Conversion to real	$y = x_r^*$ $y = x$ $y = x$ accuracy may be lost	ANSI	REAL FLOAT SINGL	R R R R R	1 1 1 1 1	C I* I R I B** D	$ x_r < 2^{46}$ $ x < 2^{23}$ $ x < 2^{46}$ $ x < \infty$ $ x < 2^{46}$ $ x < \infty$	Full	Inline
Conversion to double-precision	$y = x_r^*$ accuracy may be extended $y = x$ accuracy may be extended	ANSI	DBLE	D D D D D	1 1 1 1 1	C I* I R D B**	$ x_r < \infty$ $ x < 2^{23}$ $ x < 2^{46}$ $ x < \infty$	Full	Inline
Conversion to complex	$y = x$ $y = x_1 + i \cdot x_2$ or $y = x_1 + i \cdot 0$	ANSI	CMPLX	C C C C C	1 or 2	C I* I B** R D	$ x < \infty$ $ x_1 , x_2 < 2^{23}$ $ x_1 , x_2 < 2^{46}$ $ x_1 , x_2 < \infty$	Full	Inline
Character to integer	$y = x$	ANSI	ICHAR**	I	1	CH	any legal character	None	External
Integer to character	$y = x$	ANSI	CHAR ^S	CH CH	1 1	I B**	0-255	None	External
64-bit integer to 24-bit integer	$y = x$	CRI	INT24	I*	1 1	I B*	$ x < 2^{23}$	Full	Inline
24-bit integer to 64-bit integer	$y = x$	CRI	LINT	I	1	I*	$ x < 2^{23}$	Full	Inline

* 24-bit integer

[†] $x = x_1 + i \cdot x_2$

** Type conversion routines assign the appropriate type to Boolean arguments without shifting or manipulating the bit patterns they represent

*** ICHAR converts a character to an integer based on the character position in the collating sequence

^S CHAR(i) returns the ith character in the collating sequence

Function	Definition	ANSI/ CRI extension	Function (y)		Argument(s) (x)			Vector	Code Generated
			Name	Type	No	Type	Range		

TIME AND DATE

Real-time clock	Low order 46 bits of clock register expressed as floating point integer	CRI	RTC	R	0			None	Inline
	Current clock register content		IRTC	I	0				
Time	Current time in ASCII code (hh:mm:ss)	CRI	CLOCK [†]	B	0			None	External
Julian date	Current Julian data in ASCII code (yyddd)	CRI	JDATE [†]	B	0			None	External
Date	Current date in ASCII code (mm/dd/yy)	CRI	DATE [†]	B	0			None	External

BOOLEAN FUNCTIONS

Logical product	Bit-by-bit logical product (AND) of x_1 and x_2	CRI	AND	B	2	B,I, R,L		Full	Inline
Logical sum	Bit-by-bit logical sum (OR) of x_1 and x_2	CRI	OR	B	2	B,I, R,L		Full	Inline
Logical difference	Bit-by-bit logical difference (XOR) of x_1 and x_2	CRI	XOR	B	2	B,I, R,L		Full	Inline
Equivalence	Bit-by-bit equivalence (XOR) of x_1 and x_2	CRI	EQV	B	2	B,I, R,L		Full	Inline
Not equivalent	Bit-by-bit logical difference (XOR) of x_1 and x_2	CRI	NEQV	B	2	B,I, R,L		Full	Inline
Complement	Bit-by-bit logical complement of x_1	CRI	COMPL	B	1	B,I, R,L		Full	Inline
Mask	x_1 =number of one-bits to be left-justified if $0 < x_1 < 63$. ($128 - x_1$)=number of one-bits to be right-justified if $64 < x_1 < 128$	CRI	MASK	B	1	I		Full	Inline
Circular shift	x_1 shifts left x_2 positions; leftmost positions replace vacated positions	CRI	SHIFT	B	2	x_1 : B,I, R,L x_2 : I	$0 < x_2 < 64$	Full ^{††}	Inline

[†] These procedures can be called as subroutines also. An integer or real argument is passed to return the result. However, these procedures cannot be called as both subroutines and functions within a single program unit.

^{††} The subprogram vectorizes if the second argument is invariant (that is, x_2 is not a vector)

Function	Definition	ANSI/ CRI extension	Function (y)		Argument(s) (x)			Vector	Code Generated
			Name	Type	No	Type	Range		

BOOLEAN FUNCTIONS (cont.)

Logical shift	x_1 shifts left x_2 positions; leftmost positions lost; rightmost positions set to zero	CRI	SHIFTL	B	2	x_1 : B,I, R,L x_2 :I	$0 \leq x_2 < 64$	Full ^{††}	Inline
	x_1 shifts right x_2 positions; rightmost positions lost; leftmost positions set to zero		SHIFTR	B	2				
Leading zeros	Tallies number of leading zeros in x	CRI	LEADZ	I	1	B,I, R,L		Full	Inline
Population count	Tallies number of ones in x	CRI	POPCNT	I	1	B,I, R,L		Full	Inline ^{†††}
Population parity count	0, if x has an even number of ones 1, if x has an odd number of ones	CRI	POPPAR	I	1	B,I, R,L		Full	Inline ^{†††}
Merge	Bit-by-bit selective merge $(x_1 \wedge x_3) \vee (x_2 \wedge \neg x_3)$ [†]	CRI	CSMG	B	3	B,I, R,L		Full [§]	Inline

VECTORIZATION AIDS

Vectorization	x_1 returned if $x_3 \geq 0$ x_2 returned if $x_3 < 0$	CRI	CVMGP	B	3	B,I, R,L		Full [§]	Inline
	x_1 returned if $x_3 < 0$ x_2 returned if $x_3 \geq 0$		CVMGM						
	x_1 returned if $x_3 = 0$ x_2 returned if $x_3 \neq 0$		CVMGZ						
	x_1 returned if $x_3 \neq 0$ x_2 returned if $x_3 = 0$		CVMGN						
	x_1 returned if x_3 is true x_2 returned if x_3 is false		CVMGT			x_1, x_2 : B,I, R,L x_3 :L			

MISCELLANEOUS FUNCTIONS

Location	Returns memory address of specified variable or array	CRI	LOC	I	1	B,I,R, L,C,D		None	Inline
Tallies number of arguments used to call subprogram	y=x	CRI	NUMARG	I	0			None	Inline

[†] The logical symbol $\wedge$ represents AND, the logical symbol $\vee$ represents OR, and the logical symbol $\neg$ represents AND NOT.

^{††} The subprogram vectorizes if the second argument is invariant (that is, x_2 is not a vector).

^{†††} The CRAY-1 A and CRAY-1 B Computer Systems will be external without population parity hardware.

[§] The function cannot be passed as an argument.

The Cray FORTRAN Compiler (CFT) includes a set of utility procedures which, like intrinsic functions, is predefined by name and function. Unlike intrinsic functions, however, utility procedures are not provided for by the ANSI FORTRAN Standard. They include subroutines as well as functions; some have arguments of mixed type; and some modify these arguments' contents.

Table C-1 describes currently available utility procedures. Entities in this table are abbreviated, integer (I), real (R), double precision (D), complex (C), logical (L), Boolean (B), and Hollerith (H). Unless noted, 24-bit integer variables can be used as arguments to a function accepting integer arguments. 24-bit variables are sign extended and treated as 64-bit variables.

Notations *Full*, *Pseudo*, and *None* indicate the vector status of each subprogram. *Full* indicates the routine uses vector hardware. *Pseudo* indicates the routine uses only scalar hardware but simulates vectorization to gain efficiency for the DO-loop where such a routine appears. *None* indicates only a scalar version exists.

NOTE

Although correct argument types are specified, it is the user's responsibility to ensure that actual arguments are of the correct type. No type conversion occurs automatically.

Table C-1. CFT utility procedures

Category	Subroutine or Function Name	Func. Type	Description	Argument Type(s)			Vector	Code Generated
				x ₁	x ₂	x ₃		
TIME SYSTEM	TIMEF	R	Returns current clock register content in milli-seconds				None	External
	[†] SECOND	R	x ₁ = Cumulative CPU time for job in seconds.	R			None	External
	SYSTEM	I	Selects a COS function. See Exchange Processor, CRI publication SM-0040.	I			None	External
	EXIT		Terminates job step. Job continues at next control statement.				None	External
	SSWITCH		x ₂ = 1 if sense switch x ₁ is ON. x ₂ = 2 if sense switch x ₁ is OFF, if x ₁ < 1, or if x ₁ > 6.	I	I		None	External
	ABORT		Terminates job step after program detects error. Gives error exit to COS. Job continues at next EXIT statement. Prints traceback in logfile. Argument x ₁ is optional. If x ₁ is present, it is written to the logfile before the traceback. Argument x ₁ is subject to the same restrictions as the argument in REMARK2.	B,I, D,C,L			None	External
	ERREXIT		Terminates job step after program detects error. Gives error exit to COS. Job continues at next EXIT control statement. Prints no traceback.				None	External
	TRBK	I	Writes a traceback through subroutine calls to file x ₁ . If x ₁ is '\$LOG' or is missing, the traceback is written to the logfile.	I			None	External
	REMARK		Writes an ASCII message to both the user logfile and system logfile. The R descriptor is prohibited. For a variable or array name of type I, R, C, or D (not 24-bit integer), the caller must signal the end of the message by a null character (zero). x ₁ = 8 words maximum.	B,I, D,C,L			None	External
	REMARK2		Same as REMARK except that x ₁ may have a maximum of 9 words. The first 5 characters are a code identifying the message for machine processing.	B,I, D,C,L			None	External
	REMARKF		Enters a message in the logfile. A format and up to 12 variables can be passed, each occupying only one word. The first argument is the format label.	I,C	I,R, L,B		None	External
	SENSEFI		Determines current interrupt mode. Mode=1, interrupts permitted; Mode=0, interrupts prohibited.	I			None	External
	CLEARFI		Temporarily prohibits floating-point interrupts				None	External
	SETFI		Temporarily permits floating-point interrupts				None	External
	CLEARFIS		Permanently prohibits floating-point interrupts				None	External
	SETFIS		Permanently permits floating-point interrupts				None	External
INPUT/OUTPUT	EODW		Writes eod and, as required, eof and eor record(s) on unit x ₁	I ^{††}			None	External
	IEOF	I	Returns -1 if eod processed at unit x ₁ ; +1 if eof read at unit x ₁ ; 0, otherwise.	I ^{††}			None	External

[†] These procedures can be used either as functions or as subroutines, but not both within the same program unit.

^{††} x₁ can be a unit number or a Hollerith unit name. If no argument is specified for DUMPJOB, \$DUMP is used.

Table C-1. CFT utility procedures (continued)

Category	Subroutine or Function Name	Func. Type	Description	Argument Type(s)			Vector	Code Generated
				x ₁	x ₂	x ₃		
INPUT/OUTPUT (continued)	EOF	R	Returns -1.0 if <i>eod</i> processed at unit x ₁ ; +1.0 if <i>eof</i> read at unit x ₁ ; 0.0, otherwise.	I ^{††}			None	External
	UNIT	R	Returns -2.0 if record at unit x ₁ partly read; -1.0 if unit x ₁ transfer successful; 0.0 if <i>eof</i> or <i>eod</i> read at unit x ₁ ; +1.0 if disk parity error while reading unit x ₁ ; +2.0 if unit x ₁ error indicated. (Applies only to buffered input/output operations)	I ^{††}			None	External
	LENGTH	I	Returns number of Cray words transferred to or from unit x ₁ . Returns 0 if <i>eof</i> or <i>eod</i> read from unit x ₁ . (Applies only to buffered input/output operations)	I ^{††}			None	External
	SETPOS		Sets starting address of current record in dataset associated with unit x ₁ to beginning address of dataset if x ₂ = 0, to ending address if x ₂ = -1, or to word address x ₂ .	I ^{††}	I		None	External
DEBUGGING AIDS	ENDRPV		Resumes error exit processing suspended by SETRPV				None	External
	SETRPV		Transfers control to a specified routine upon encountering a user-selected system error. x ₁ = Name of subroutine to which control is passed. x ₂ = Array to receive the Exchange Package and abort conditions. x ₃ = Mask defining the error class from which reprieve is desired.	H	I,R,L	I,B	None	External
	SYMDEBUG		Dumps the contents of specified program variables. The character string argument is identical to the parameters for the DEBUG control statement. See the CRAY-OS Version 1 Reference Manual, publication SR-0011.	H			None	External
	DUMPJOB		Creates an unblocked dataset containing the user job area image (including register states) suitable as input to the DUMP programs	I ^{††}			None	External

† These procedures can be used either as functions or as subroutines, but not both within the same program unit.

†† x₁ can be a unit number or a Hollerith unit name. If no argument is specified for DUMPJOB, \$DUMP is used.

The Cray FORTRAN compiler (CFT) produces five types of compile-time messages. From least severe to most severe, they are

COMMENT	-	Message reports a programming inefficiency.
NOTE	-	This usage may cause problems with other compilers.
CAUTION	-	Message reports a possible user error.
WARNING	-	Message reports a probable user error.
ERROR	-	Message reports a fatal error.

CFT produces these messages in the following general format.

severity type	message	location number
---------------	---------	-----------------

(Location number is the internal CFT location where the message was generated.)

If a message in this appendix is preceded by \$, the print position at which the error occurred is specified by the following.

NEAR >> <<

Error messages can occur under the following conditions.

- After the first non-declarative statement - An error message occurring after the first non-declarative statement usually indicates contradictions between declarative statements.
- After an END statement - An error message occurring after an END statement usually refers to missing statement or format numbers.
- After any other statement - An error message occurring after any other statement usually refers to a syntax error in that statement. If no source list is being generated, the first line of the statement is listed. Continuation lines, if any, are not listed but are indicated by four plus signs appended to the first line of the statement.

Logfile messages follow the compile-time messages in this appendix.

COMPILE-TIME MESSAGES

"name" ALREADY SAVED AT SEQUENCE NUMBER *n*

"name" is in a SAVE statement list and the preceding SAVE statement list at sequence number *n*.

"name" APPEARS TWICE IN DUMMY ARGUMENT LIST

A symbolic name appears twice in the dummy argument list of a FUNCTION or SUBROUTINE statement.

"name" CANNOT BE DECLARED EXTERNAL

"name" has appeared in both an EXTERNAL statement and in an array declarator, DATA, COMMON, or PARAMETER statement.

"name" CANNOT BE DECLARED INTRINSIC

The symbolic name was used in the program unit, giving the name a type; for example, a dummy argument.

"name" DOUBLY ASSOCIATED IN EQUIVALENCE AT SEQUENCE "*number*"

EQUIVALENCE statements using "name" are incorrect and specify an illegal storage sequence. To correct the error, the following two situations must be true. The same storage unit cannot occur more than once in a storage sequence. The example

```
REAL A(2)
EQUIVALENCE (X,A(1)), (X,A(2))
```

is incorrect because A(1) and A(2) have the same storage unit. Consecutive storage units cannot be specified as nonconsecutive. The example

```
REAL A(2)
DOUBLE D(2)
EQUIVALENCE (A(1),D(1)), (A(2),D(2))
```

is incorrect because A(1) and A(2) are nonconsecutive.

"name" IS STATICALLY ALLOCATED

CFT assigns "name" to static storage; otherwise, the name can be removed from SAVE and DATA statements.

"name" NO LONGER INTRINSIC

An intrinsic function name appeared in a compiler directive, giving external subprograms special attributes (for example, VFUNTION and NO SIDE EFFECTS). The intrinsic function loses its intrinsic properties. The intrinsic function should appear in an EXTERNAL statement before the compiler directive declarations.

"name" NOT AN INTRINSIC FUNCTION

A symbolic name in the list of an INTRINSIC statement is not an intrinsic function.

"name" NOT LOCAL OR COMMON BLOCK VARIABLE

List may contain only local or common block variables.

ADJUSTABLE DIMENSION ILLEGAL IN MAIN PROGRAM

Arrays in main program must have constant subscripts.

ARGUMENTS IGNORED FOR ZERO ARGUMENT INTRINSIC

An intrinsic function defined to have zero arguments was referenced with one or more arguments. Reference zero argument intrinsics with a null argument list.

ARRAY NAMED FORMAT IS POTENTIALLY AMBIGUOUS

An array with the symbolic name 'FORMAT' was declared using the DIMENSION, COMMON, INTEGER, REAL, DOUBLE, or COMPLEX statement.

ASSUMED SIZE DIMENSION ILLEGAL IN MAIN PROGRAM

Arrays in main program must have constant subscripts.

AT SEQUENCE NUMBER n : "name" CANNOT BE SAVED

"name" is in a SAVE statement list at sequence number n , but "name" cannot appear in a SAVE statement list.

BAD CONSTANT LIST IN DATA STATEMENT

A DATA statement is missing a constant list or contains an illegal separator character.

BAD REPETITION FIELD

The repetition count in a DATA constant list must be an integer greater than 0.

§BAD STATEMENT FUNCTION PARAMETER LIST

The formal parameter list in an arithmetic statement function definition statement contains an illegal element.

BAD SUBSCRIPT IN DATA STATEMENT

A subscript must be an integer constant or constant name in a DATA statement.

BAD TRIP COUNT IN IMPLIED DO

Incrementation parameter m_3 has been assigned a value of 0 or $(m_2 - m_1 + m_3)/m_3$ is negative or 0.

CALL OF NON EXTERNAL FUNCTION "name"

Called external procedure does not exist or a name has been used for both a variable and an external procedure.

§CHARACTER COUNT TOO LARGE

An R-form Hollerith constant is specified with more than eight characters or an H- or L-form Hollerith constant is specified with more than eight characters in other than a DATA statement or an actual argument list.

CHARACTER LENGTH MUST BE >ZERO and <505

A character entity must be assigned a length greater than 0 and less than 505.

COMMA EXPECTED

A required comma has not been specified.

COMMA EXPECTED IN EQUIVALENCE AT SEQUENCE "*number*"

A required comma has been omitted in an EQUIVALENCE statement.

COMMA OR RIGHT PARENTHESIS EXPECTED

A required comma or right parenthesis was omitted.

COMMA OR RIGHT PARENTHESIS EXPECTED IN EQUIVALENCE AT SEQUENCE "*number*"

A required comma or right parenthesis was omitted in an EQUIVALENCE statement.

COMPILER ERROR

The CFT compiler detected an error in its internal tables; please show your listing to a Cray Research systems analyst.

COMPILER ERROR - INTERNAL TABLE OVERFLOW, RECOMPILE WITH SMALLER BLOCK SIZE

A CFT internal table has overflowed when compiling. The optimization block where the error occurs must be reduced in size by inserting a CDIR\$ BLOCK compiler directive in the block or reducing the maximum block size with the MAXBLOCK control statement parameter.

CONFLICTING TYPE FOR INTRINSIC FUNCTION "*name*" IGNORED

A type statement cannot change the type of an intrinsic function. The function must be declared EXTERNAL before its type can be changed.

CONFLICTING USE OF INTRINSIC FUNCTION "*name*"

An intrinsic function name has been used to reference the function in another way within the program unit.

CONSTANT DIMENSION TOO LARGE

All dimensions must be less than 2^{22} .

CONSTANT LIST LONGER THAN VARIABLE LIST

Constants and variables in a DATA statement must have a one-to-one correspondence.

§CONSTANT SUBSCRIPT TOO LARGE

Statement contains a subscript which, when evaluated, yields a result greater than the size of the named array.

CONTROL LIST MUST INCLUDE ONE FILE OR UNIT OPTION

INQUIRE statement must specify either a file or a unit.

CONTROL LIST MUST INCLUDE ONE UNIT OPTION

The I/O statement must specify a unit.

\$DATA ENTRY IN BLANK COMMON ILLEGAL

The DATA statement cannot be used to initialize blank common.

DECLARATOR "name" MUST BE DUMMY ARGUMENT OR IN COMMON

The dimension declarator "name" in an adjustable array declarator was not a dummy argument or a variable in common.

DEPENDENCY INVOLVING ARRAY "name"

A dependency exists involving two array references in the same statement. This dependency inhibits DO-loop vectorization.

DEPENDENCY INVOLVING ARRAY "name" IN SEQUENCE NUMBER nnnnnnnnn

A dependency exists involving an array reference in sequence number nnnnnnnnn and an array reference in the AT SEQUENCE NUMBER n line preceding this message. This dependency inhibits DO-loop vectorization.

DIMENSION COUNT > SEVEN

More than seven dimensions appear in an array declarator.

DIMENSION EXCEEDED

A subscript in a DATA statement element exceeds the corresponding array declaration.

DIRECTIVE NO LONGER SUPPORTED

The SCHED and NOSCH directives are no longer supported. The directives have no effect if used.

DIVIDE BY ZERO

Dividing by the constant 0 is illegal.

DO ILLEGAL ON CONDITIONAL STATEMENT

This type of statement is not allowed as the conditional statement of a direct logical IF statement.

DO INDEX ACTIVE

The loop control variable is already active from a previous loop.

DO INDEX IN INPUT LIST

Attempt to read data into a DO variable.

DO LOOP MAY NOT CROSS BLOCK BOUNDARY

A DO-loop that begins within an IF-block, ELSE-block or ELSE IF-block must be totally contained within that block. A block that begins within a DO-loop must be totally contained within the loop.

DO TERMINATOR ILLEGAL IN CONDITIONAL BLOCK STATEMENT

DO-loop must not terminate on an IF (e) THEN, ELSE, ELSE IF, or ENDIF statement.

DO TERMINATOR PRECEDES DO STATEMENT

The statement label that terminates a DO-loop precedes the corresponding DO statement.

DOUBLY DEFINED FUNCTION OR MISSING DIMENSION

An arithmetic statement function is defined more than once or an array was not dimensioned.

DOUBLY DEFINED STATEMENT NUMBER

Statement labels cannot be defined more than once in a program unit.

DUMMY ARGUMENT IN EXECUTABLE STATEMENT PREVIOUS TO ENTRY

A dummy argument name in an executable statement must also be specified in the FUNCTION, SUBROUTINE, or ENTRY statement referenced before the executable statement.

DUPLICATE COMMON DEFINITION "*name*"

Variable or array appears in common more than once.

DUPLICATE CONTROL OPTION IN LIST

An option is specified more than once in an I/O statement control list.

DUPLICATE DIMENSION "*name*"

Dimensions cannot be declared more than once.

DUPLICATE TYPE DEFINITION "*name*"

Variables cannot be given more than one type. This message is issued when a type statement redefines the variable type with a type established by a DATA, POINTER, or previous type statement.

DYNAMIC BLOCK "*name*" NOT IN PREVIOUS COMMON

Dynamic name must be declared as a common block previous to its appearance in a DYNAMIC compiler directive.

\$EBCDIC NOT IMPLEMENTED

The current version of CFT allows only ASCII characters.

ENCODE/DECODE MAY NOT BE LIST DIRECTED

The format identifier in an ENCODE or DECODE statement must not specify list-directed I/O; it cannot be an *.

ENTRY NAME ILLEGAL

ENTRY name not a function or subroutine name.

ENTRY "*name*" USED AS DUMMY ARGUMENT

"*name*" is an entry point (that is, "*name*" appeared in a FUNCTION, SUBROUTINE, ENTRY, or BLOCK DATA statement) and appears as a dummy argument.

ENTRY STATEMENT ILLEGAL IN DO LOOP OR BLOCK IF

The ENTRY statement must not be used in a DO-loop or a block IF.

ENTRY STATEMENT ILLEGAL IN MAIN PROGRAM

The ENTRY statement must not be used in a main program. It is used only in a subroutine or function.

SEQUIVALENCE EXTENDS COMMON BLOCK BASE

Common block storage is illegally extended by adding storage units preceding the first storage unit specified in the COMMON statement.

EQUIVALENCE OF "name" IN DIFFERENT COMMON BLOCKS

An EQUIVALENCE statement must not associate the storage sequences of two different common blocks in the same program unit.

ERROR IN CONSTANT

Illegal characters in constant, or constant out of range

SEXPRESSION ILLEGAL IN INPUT LIST

Input list item is not a variable name, array element name, or array name.

EXPRESSION TYPE MUST BE INTEGER

Expressions in alternate RETURN statement must be type integer.

SEXTRA CHARACTERS AFTER END OF STATEMENT

Characters are specified after the syntactic end of a statement.

SEXTRA CHARACTERS AFTER END OF STATEMENT IN EQUIVALENCE AT SEQUENCE "number"

Characters are specified after the syntactic end of an EQUIVALENCE statement.

SEXTRA COMMA OR MISSING PARAMETER

Either the statement contains an extra comma or a parameter or list item has been omitted.

FEWER SUBSCRIPTS USED THAN DECLARED

A reference to an actual array element has fewer subscript expressions in its subscript than dimension declarators in the corresponding array declarator. The missing subscript expressions are assumed rightmost in the subscript and are each assigned the value 1 by the compiler.

FORMAT ERROR

FORMAT statement contains an editing error.

FORMAT MUST BE CHARACTER EXPRESSION

A FORMAT specifier can be an expression only if the expression is a character expression,

FUNCTION "name" ALREADY DECLARED EXTERNAL

The symbolic name appearing in an INTRINSIC statement has already appeared in an EXTERNAL statement.

FUNCTION "name" ALREADY DECLARED INTRINSIC

The symbolic name appearing in an INTRINSIC statement has already appeared in an INTRINSIC statement.

FUNCTION "*name*" MORE THAN 6 CHARACTERS

An attempt was made to declare a function with a name greater than six characters as having a vector call-by-value version (CDIR\$ VFUNCTION). Rename the function with a shorter name.

FUNCTION "*name*" MUST BE DECLARED IN INTRINSIC OR EXTERNAL STATEMENT

A function passed to another subprogram as an actual argument must be declared in an INTRINSIC statement (intrinsic functions) or an EXTERNAL statement (user-supplied functions).

FUNCTION "*name*" NOT DECLARED

One of the following conditions exists with IMPLICIT NONE or IMPLICIT SKOL specified.

1. Function "*name*" has not appeared in an EXTERNAL statement.
2. "*name*" was intended to be an array but did not appear in an array declarator.

FUNCTION OR CALL "*name*" REFERENCES ITSELF

A reference to the function or subroutine subprogram being compiled is encountered with that subprogram.

$S_{FUNCTION}$ USED WITH INCORRECT NUMBER OF ARGUMENTS

The number of arguments in the function reference does not agree with the number of arguments in the function definition.

GROUP NAME DEFINED PREVIOUS TO NAMELIST

A group name may be defined only in NAMELIST.

$S_{H,L,R}$ COUNT < OR = ZERO

In an nH , nL , or nR specification of a Hollerith value, n is less than or equal to 0.

$S_{H,L,R}$ COUNT PAST END OF STATEMENT

In an nH , nL , or nR specification of a Hollerith value, n specifies more characters than are provided, or an apostrophe terminating a Hollerith string is missing.

$S_{HOLLERITH}$ CONSTANT > EIGHT CHARACTERS

A Hollerith constant of more than eight characters is specified in other than H- or L-form and in other than an actual argument list or a DATA statement constant.

IF BLOCK LEVEL NOT = ZERO AT END STATEMENT

An ENDIF statement is missing.

ILLEGAL BY VALUE CALL

A by-value function call requires more than seven S or V registers to pass its arguments. A by-value call cannot use more than seven registers. Reduce the register number to a number less than eight or pass the arguments by address instead of by value.

SILLEGAL CHARACTER

A nonstandard FORTRAN character, misplaced character, or syntax error has been encountered.

ILLEGAL CHARACTER EXPRESSION

A character assignment expression (for example, $v=e$) is illegal because v is used in expression e .

SILLEGAL CHARACTER OR MISSING DIMENSION

Either the statement contains an illegal character or an array element has not been defined by a DIMENSION statement.

SILLEGAL CHARACTERS IN NAME FIELD

Illegal characters are in a field that must contain a symbolic name.

SILLEGAL CHARACTERS IN NAME FIELD IN EQUIVALENCE AT SEQUENCE "number"

Illegal characters are in an EQUIVALENCE field that must contain a symbolic name.

SILLEGAL CHARACTERS IN STATEMENT NUMBER FIELD

Non-numeric characters appear in what should be a numeric field.

SILLEGAL COMMON BLOCK NAME

The specification of a common block name does not conform to the rules for constructing symbolic names.

SILLEGAL COMPILER DIRECTIVE

C DIR\$ omitted or misspelled in columns 1 through 5, 6 not blank or zero, compiler directive not in columns 7 through 72, or compiler directive is misspelled or does not exist for CFT.

ILLEGAL COMPLEX RELATION

Relational operator for complex operands must be .EQ. or .NE.

ILLEGAL CONDITIONAL STATEMENT

The conditional statement go logical IF must not be a logical IF statement or a block statement IF.

ILLEGAL CONTINUATION

More than 19 consecutive continuation cards encountered or the first card of a program unit is a continuation card.

ILLEGAL CONTROL OPTION

An option in the control list of an I/O statement is incorrect.

ILLEGAL CONVERSION IN DATA STATEMENT

The types of a variable and an associated constant in a DATA statement differ. The type conversion required is illegal or undefined.

SILLEGAL DO INDEX

DO-variable is not an integer, real, or double-precision variable.

SILLEGAL DO TERMINATOR

DO-loops must not terminate on unconditional transfer statements.

ILLEGAL DO VARIABLE OR PARAMETER TYPE

The DO-loop variable or parameter is not type integer, real, double-precision, or Boolean.

SILLEGAL FORMAT NAME

A format identifier cannot be recognized as a statement label or the name of an array.

SILLEGAL IMPLICIT STATEMENT ARGUMENTS

IMPLICIT statement argument is not an alphabetic character or the range of characters specified is illegal.

ILLEGAL LOGICAL RELATION

A logical variable must not be used in a relational expression.

ILLEGAL MIX OF CHARACTER AND NONCHARACTER IN COMMON BLOCK "name"

It is illegal to mix character and noncharacter entities in the same common block.

ILLEGAL MIX OF CHARACTER AND NONCHARACTER IN EQUIVALENCE AT SEQUENCE "number"

It is illegal to mix character and noncharacter entities in the same EQUIVALENCE statement.

ILLEGAL MIXED MODE OR CONVERSION

The types of two operands in an expression are incompatible or the type of array element or variable being defined is incompatible with the type of expression being evaluated.

SILLEGAL NUMBER IN NAME FIELD

A symbolic name must not begin with a number.

SILLEGAL OR DUPLICATE PARAMETER DEFINITION

Symbolic name of type integer, real, double precision, or complex not followed by an arithmetic expression. Symbolic name of type logical not followed by a logical expression. A symbolic name has been assigned more than once in the same program unit.

ILLEGAL PLACEMENT OF ALIGN, DIRECTIVE IGNORED

The ALIGN compiler directive was not placed immediately before a DO statement, a statement with a referenced statement label, a PROGRAM statement, a SUBROUTINE statement, a FUNCTION statement, or an ENTRY statement. The directive will be ignored.

ILLEGAL POINTEE "*name*"

A pointee cannot be a dummy argument or a pointer. It cannot be equivalenced or be specified in a common block statement.

ILLEGAL POINTER VARIABLE "*name*"

A pointer must be a simple variable. It cannot appear in an EQUIVALENCE statement. If defined in a PARAMETER or DATA statement, the definition must not precede its definition as a pointer.

ILLEGAL STATEMENT LABEL IN IO CONTROL LIST

A 1- to 5-digit statement number is missing after END= or ERR=.

ILLEGAL STATEMENT SEQUENCE

An improper sequence of statement types has been encountered (for example, a GO TO statement followed by a DIMENSION statement).

ILLEGAL STATEMENT TYPE

A statement keyword is misspelled (for example, DIMENSOIN) or is otherwise unidentifiable.

ILLEGAL SUBSCRIPT TYPE "*name*"

A subscript expression is not of type integer or contains a constant that exceeds $2^{24}-1$.

ILLEGAL SUBSTRING

A substring for a character item is incorrectly formed or an attempt is made to use a substring with an entity which can not have a substring (such as a character constant).

ILLEGAL SYNTAX IN NAMELIST

Illegal element found in NAMELIST statement.

ILLEGAL TYPE FOR ASSIGNED VARIABLE

A variable reference in an ASSIGN statement is not of type integer.

$\S$ ILLEGAL TYPE LENGTH

Length specified is not allowed for this data type.

ILLEGAL UNIT SPECIFIER

The unit specifier for INQUIRE must be an integer expression.

$\S$ ILLEGAL USE OF ** IN CONSTANT EXPRESSION

A constant expression specifies exponentiation to a non-integer power.

ILLEGAL USE OF ASSUMED CHARACTER LENGTH

A character entity with a length of * must be a dummy argument, the symbolic name of a constant, or an external function whose name appears in a FUNCTION or ENTRY statement within the same program unit.

ILLEGAL USE OF ASSUMED SIZE ARRAY "*name*"

An array with an asterisk for the last dimension cannot be used without subscripts in an I/O statement.

ILLEGAL USE OF COLON

A colon can only be used in a FORMAT statement or to separate the lower and upper dimensions in a declarative.

ILLEGAL USE OF DUMMY ARGUMENT "name"

A dummy argument in a procedure subprogram cannot be named the same as a local variable or another dummy argument.

ILLEGAL USE OF DUMMY ARGUMENT "name" IN EQUIVALENCE AT SEQUENCE "number"

Dummy arguments may not appear in an EQUIVALENCE statement.

ILLEGAL USE OF FUNCTION "name"

A function name cannot be used as an array name.

ILLEGAL USE OF FUNCTION "name" IN EQUIVALENCE AT SEQUENCE "number"

A function name cannot be used as an array name in an EQUIVALENCE statement.

ILLEGAL USE OF "name" IN I/O LIST

External, function, or program name not permitted in an I/O list.

ILLEGAL USE OF "name"

Group name referenced previous to its definition in a NAMELIST statement.

ILLEGAL USE OF NAMELIST GROUP "name"

A namelist group "name" can be used only as a group name in a NAMELIST read or write.

ILLEGAL VALUE IN CONSTANT EXPRESSION

The evaluation of a constant expression yields a result that is out of range.

IMPLICIT NONE MUST BE ONLY IMPLICIT STATEMENT

IMPLICIT NONE or IMPLICIT SKOL appear in the same program unit as another IMPLICIT statement.

IMPROPERLY NESTED DO LOOP

Inner DO-loop is not contained entirely within the outer DO-loop range.

INCORRECT ARGUMENT TYPE

Actual argument is of the wrong type in a function reference.

INPUT FILE EMPTY

An end-of-file record was encountered as the first record of the source input dataset.

INTEGER*2=24-BIT INTEGER

INTEGER*2 is implemented as a 24-bit integer by CFT.

INTEGER CONSTANT EXPRESSION REQUIRED

The subscript or substring expression is not an integer constant expression.

INTRINSIC FUNCTION "*name*" CANNOT BE ACTUAL ARGUMENT

Certain intrinsic functions cannot be passed to subprograms as actual arguments.

INTRINSIC FUNCTION TYPE USED WITH ILLEGAL ARGUMENT

The actual argument(s) to the intrinsic function is an improper type.

IO CONTROL LIST SPECIFIER MUST BE CHARACTER EXPRESSION

The I/O control list specifier must be evaluated to a character value.

IO CONTROL LIST SPECIFIER MUST BE CHARACTER VARIABLE OR ARRAY ELEMENT

The I/O control list specifier can be a character variable or an array element.

IO CONTROL LIST SPECIFIER MUST BE INTEGER EXPRESSION

The I/O control list specifier must be evaluated to an integer value.

IO CONTROL LIST SPECIFIER MUST BE INTEGER VARIABLE OR ARRAY ELEMENT

The I/O control list specifier can be an integer variable or an array element.

IO CONTROL LIST SPECIFIER MUST BE LOGICAL VARIABLE OR ARRAY ELEMENT

The I/O control list specifier can be a logical variable or an array element.

LAST ARRAY NOT FILLED

The last element in a DATA statement variable list is an unsubscripted array and not enough constants are specified to completely fill the array. Remaining elements of the array are not initialized.

LEFT PARENTHESIS EXPECTED

A required opening parenthesis was omitted.

LEFT PARENTHESIS EXPECTED IN EQUIVALENCE AT SEQUENCE "*number*"

A required opening parenthesis was omitted in an EQUIVALENCE statement.

LINE LENGTH > 133 CHARACTERS

One or more lines exceeds 133 characters during FORMAT statement editing.

LIST DIRECTED IO ILLEGAL FOR INTERNAL FILE

List-directed reads and writes all illegal operations on internal files. Internal file IO must be formatted.

LOGICAL OPERATOR MUST END IN PERIOD

A period does not follow an otherwise correct logical operator.

LOSS OF PRECISION IN TYPE CONVERSION

The type of a variable and the type of the associated constant in a DATA statement differ. The constant is converted to the type of the variable and precision is lost.

MAXIMUM LEGAL ITERATION COUNT EXCEEDED

A DO-loop trip count is larger than the allowable maximum of $2^{23}-1$.

\$MISSING =

An equal sign is missing in a PARAMETER or statement function definition statement.

MISSING = IN CONTROL LIST

There is no equal sign after an option in an I/O statement control list.

MISSING COLON

A required colon has been omitted in a substring expression.

MISSING END STATEMENT

The last or only program unit being compiled lacks an END statement in its last line.

MISSING OR ILLEGAL CONSTANT LIST

A PARAMETER or DATA statement has not specified a constant list, or a list has a missing separator.

\$MISSING OR ILLEGAL STATEMENT NUMBER IN DO

The statement number is missing or it contains illegal characters in a DO statement.

MISSING STATEMENT NUMBER IN ASSIGN

An ASSIGN statement lacks a statement label reference.

MISSING TO IN ASSIGN STATEMENT

An ASSIGN statement requires the keyword extension TO.

MORE THAN 312 DUMMY ARGUMENTS IN PROGRAM UNIT

CFT does not accept more than 312 dummy arguments in a subroutine or function subprogram. Each argument for an entry point in a program unit represents a separate argument when computing the number of arguments used in a program unit.

MORE THAN ONE ELSE STATEMENT AT THIS IF LEVEL

Only one ELSE statement is permitted per IF-level.

\$NAME LONGER THAN EIGHT CHARACTERS

A symbolic name must not contain more than eight characters.

NO BLOCK IF ASSOCIATED WITH ELSE STATEMENT

An ELSE statement must follow a block IF statement and precede an END IF statement of the same level.

NO BLOCK IF ASSOCIATED WITH END IF STATEMENT

An END IF must be uniquely associated with an IF(*e*) THEN statement of the same IF-level.

NO PATH TO THIS STATEMENT

The previous statement is an unconditional transfer and this statement has no statement number.

NONSTANDARD STATEMENT

The statement is a non-ANSI FORTRAN statement supported by CFT.

NOT ENOUGH DO PARAMETERS

Fewer than two arguments have been encountered after the equal sign in a DO statement.

NOT ENOUGH MEMORY TO COMPILE

The program unit is too long to compile in the available memory.

OPTIMIZATION BLOCK BROKEN AT THIS POINT

The code size forced CFT to terminate an optimization block at this point. A new optimization block begins with the next statement.

PARAMETER USED TWICE IN STATEMENT FUNCTION PARAMETER LIST

A given symbolic name can appear only once in a single dummy argument list.

PLEASE RERUN WITH SMALLER VALUE FOR MAXBLOCK

This message follows a compiler error or an internal compiler error message if the value for the MAXBLOCK control statement parameter is greater than the system default value.

POINTER MUST BE TYPE INTEGER

A pointer variable must not be assigned a type other than integer.

POSSIBLE BRANCH TO INACTIVE DO LOOP; STATEMENT LABEL "*nn*"

CFT detected a branch to a labeled statement inside the range of a DO-loop from a branch statement outside the range of the DO-loop.

POSSIBLE BRANCH TO INACTIVE DO LOOP VIA ASSIGN OR END=/ERR= WITH LABEL "*nn*"

"*nn*" is a statement label defined within the range of a DO-loop. It has appeared in an ASSIGN statement or in the END= or ERR= branch of an I/O statement in the program unit. Verify that the branches to the statement label occur only within the innermost DO-loop where the label is defined.

PREVIOUS IMPLICIT REFERENCES THIS CHARACTER

Only one IMPLICIT reference is permitted per character.

PREVIOUS REFERENCES TO "*name*"

An ENTRY name has been used before its declaration as an ENTRY.

PROGRAM UNIT TOO LARGE TO COMPILE

One of CFT's internal tables has overflowed because there is too much code in a program unit.

REAL*8 = SINGLE PRECISION

REAL*8 is implemented as single-precision by CFT.

REFERENCES TO ARRAY "*name*" WITH NO SUBSCRIPTS

The array named was referenced without subscripts in a statement that required them.

RETURN ILLEGAL IN MAIN PROGRAM

A RETURN statement is encountered in a main program unit.

RIGHT PARENTHESIS EXPECTED

A required closing parenthesis was omitted.

RIGHT PARENTHESIS EXPECTED IN EQUIVALENCE AT SEQUENCE "*number*"

A required closing parenthesis was omitted in an EQUIVALENCE statement.

SCALAR DUMMY ARGUMENT "*name*" USED AS FORMAT IDENTIFIER

The integer variable named appears both as a format identifier and as an entry in a dummy argument list in this program unit. The cause might be a missing DIMENSION statement.

SCAN STOPPED, TOO MANY ERRORS IN FORMAT

CFT attempts recovery of up to three errors before abandoning the FORMAT statement.

SPECIFIER RECL LEGAL IF AND ONLY IF ACCESS IS DIRECT

In an OPEN statement, the RECL control list option is supported only if direct access is specified.

STATEMENT FUNCTION "*name*" IN COMMON OR ARGUMENT LIST

Statement function must not appear as a variable in a common block or an argument list.

STATEMENT FUNCTION "*name*" REFERENCES ITSELF

A statement function definition statement cannot be recursive.

§ STATEMENT FUNCTION PARAMETER MUST NOT BE ARRAY

The names of variables appearing as dummy arguments of a statement function have a scope of that statement only.

STATEMENT LABEL IGNORED

Statement label is ignored because transfer to this statement is prohibited.

STATEMENT LENGTH EXCEEDED

The statement, when arithmetic statement functions have been expanded, exceeds CFT's limit on size of statements.

STATEMENT NUMBER ILLEGAL ON DECLARATIVE

CFT does not allow a statement number on ENTRY statements.

STATEMENT NUMBER ON BLANK LINE IGNORED

Blank lines cannot contain statement labels.

SUBROUTINE "name" NOT DECLARED

IMPLICIT NONE or IMPLICIT SKOL has been specified but "name" did not appear in an EXTERNAL statement.

SUBSCRIPT OUT OF DIMENSION BOUNDS IN EQUIVALENCE AT SEQUENCE "number"

Subscript exceeds the value given in the dimensions.

SUBSTRING EXPRESSION OUT OF BOUNDS

In a substring expression "(C1:C2)", the relations $1 \leq C1 \leq C2$
 $\leq \text{LEN}$ do not all hold (where LEN is the declared length of the character entity).

SYNTAX ERROR

Illegal element, name where number required, or extra or missing punctuation.

SYNTAX ERROR IN ENCODE OR DECODE STATEMENT

Illegal element in ENCODE or DECODE statement.

§SYNTAX ERROR IN IMPLIED DO

An implied-DO list specified in a DATA statement is of improper syntactical form, references a variable that is not an implied-DO variable, or references an array element that does not specify the implied-DO variable for this implied-DO list in its subscript.

SYNTAX ERROR IN IO CONTROL LIST

Illegal element in I/O control list.

TEST EXPRESSION MUST BE LOGICAL

Expression type in a logical IF must be logical or Boolean.

TEST EXPRESSION MUST NOT BE CHARACTER

Cannot have character expression type in a logical or arithmetic IF statement.

TEST EXPRESSION MUST NOT BE LOGICAL

Expression type in an arithmetic IF must not be type logical.

TOO MANY COMMON BLOCKS DECLARED

More than 120 distinct common blocks were declared in a single program unit.

TOO MANY DO PARAMETERS

More than three arguments have been encountered after the equal sign in a DO statement.

TOO MANY DO-LOOPS ON STATEMENT

More than 15 DO-loops ended on the same statement.

TOO MANY POINTERS DECLARED

More than 312 pointers were declared in a single program unit.

TOO MANY SUBSCRIPTS

An array reference contains more subscripts than the subscripts declared.

TOO MANY SUBSCRIPTS IN EQUIVALENCE AT SEQUENCE "*number*"

An array reference in an EQUIVALENCE statement has more subscripts than were declared.

TYPE CONVERSION IN DEFINITION

A constant in a PARAMETER statement was not converted to the type of the corresponding symbolic name.

TYPE OF "*name*" NOT DECLARED

"*name*" was declared in an EXTERNAL statement, but did not appear in an explicit type statement.

TYPE STATEMENT IGNORED FOR INTRINSIC FUNCTION "*name*"

Type statements do not change the type of an intrinsic function and are ignored.

UNBALANCED PARENTHESIS

Opening and closing parentheses do not match; required parenthesis not present.

UNDEFINED ITEM IN CONSTANT EXPRESSION

A constant expression in a PARAMETER or DATA statement is specified with other than constants or the symbolic names of constants. A constant expression in a DATA statement is specified with other than constants, the symbolic names of constants, or the names of implied-DO variables.

UNDEFINED STATEMENT NUMBER "*number*"

A referenced statement label is not defined.

UNEXPECTED END OF STATEMENT

A statement encountered is syntactically incomplete.

UNIT=* ILLEGAL FOR DIRECT ACCESS

[UNIT=]* appeared in a direct access READ or WRITE statement.

UNIT=* ILLEGAL FOR UNFORMATTED IO

[UNIT=]* appeared without a format identifier in a READ or WRITE statement.

UNIT=* LEGAL ONLY IN READ OR WRITE

[UNIT=]* appeared in an auxiliary I/O statement.

UNKNOWN LOGICAL OPERATOR

The characters following a period do not represent a logical operator.

UPPER DIMENSION < LOWER DIMENSION

The lower dimension must be less than or equal to the upper dimension.

USE OF END ILLEGAL IN WRITE CONTROL LIST

END= may not be specified in a WRITE statement.

VALUE NOT ASSIGNED TO FUNCTION NAME

Function subprogram is missing value assignment for the function.

VARIABLE DIMENSION ARRAY "*name*" MUST BE DUMMY ARGUMENT

A variably dimensioned array must appear as a dummy argument at some entry point.

VARIABLE DIMENSION ILLEGAL FOR ARRAY IN COMMON

An attempt was made to put a variably dimensioned array into COMMON.

VARIABLE LIST LONGER THAN CONSTANT LIST

Constants and variables must correspond one-to-one in a DATA statement.

VARIABLE "*name*" USED AS ARRAY OR FUNCTION

A simple variable is referenced with either subscripts or an argument list.

ZERO SUBSCRIPT INCREMENT

A CII subscript must have a nonzero increment.

ZERO TO NEGATIVE POWER

Raising zero to a zero or negative power produces unpredictable results in an executable program.

LOGFILE MESSAGES

The following messages appear in the logfile following the CFT statement if the indicated condition occurs. The conditions cause compiler execution to terminate after processing the CFT statement. Control statement processing resumes with an EXIT statement if there is one in the control statement file; otherwise, the job terminates.

CF007 - BAD PARAMETER TO KEYWORD *keyword* = *parameter*

The parameter for the keyword is out of range or undefined.

CF008 - NULL INPUT FILE ILLEGAL

I=0 is an illegal input.

- CF009 - B=0 and ON=Z INCOMPATIBLE OPTIONS
B must specify a file if the Z option is on.
- CF010 - ON = *character* PARAMETER NOT ALPHA
CF010 - OFF = *character* PARAMETER NOT ALPHA
All characters in the strings for ON and OFF must be alphabetic.
- CF011 - *letter* OPTION NOT IMPLEMENTED
No existing ON/OFF option is associated with the letter.
- CF012 - *nn* CFT CONTROL CARD ERRORS
Gives count of control card errors.
- CF013 - WARNING: *string* WILL BE SET TO OFF
Options listed in *string* appear in both the ON= and OFF= keyword parameter lists.
- CF014 - DOUBLY DEFINED OPTION FOR OPT= KEYWORD
An option set by an OPT= keyword parameter was defined two times or redefined in the parameter list.
- CF015 - HEAP BASED ALLOCATION NOT YET IMPLEMENTED
ALLOC=HEAP was specified on the CFT control statement. Heap Memory management is not implemented by CFT.
- CF016 - CPU TYPE UNKNOWN - MAY EFFECT GENERATED CODE
The CPU type from the Job Communication Block is unknown to CFT and optimizations may be affected.
- CF017 - 1 WARNING
CF017 - *n* WARNINGS
Warning errors were encountered during compilation.
- CF018 - WARNING: MAXBLOCK WILL BE SET TO 1
When compiling with DEBUG on the CFT control card, MAXBLOCK is set to 1.
- CF019 - WARNING: Z WILL BE SET TO ON
When compiling with DEBUG on the CFT control card, Z is forced on.
- CF020 - WARNING: I WILL BE SET TO ON
When compiling with DEBUG on the CFT control card, I is forced on.
- CF021 - M NOT SUPPORTED: REMOVE BY 1.14
The CFT ON=M option is no longer supported. The directive is ignored.
- CF022 - Y NOT SUPPORTED: REMOVE BY 1.14
The CFT ON=Y option is no longer supported. The directive is ignored.

INFORMATIVE DEPENDENCY MESSAGES

When a dependency message is issued, another message also appears explaining why the dependency exists. The following list contains all the informative messages with examples of DO-loops causing the message to be issued. Examples of the first two messages can be found in part 3, section 2.

PREVIOUS PLUS WITH A DECREMENTING SUBSCRIPT

PREVIOUS MINUS WITH AN INCREMENTING SUBSCRIPT

POTENTIAL PROBLEM WITH EQUIVALENCED ARRAYS

```
DIMENSION E(100),D(50)
EQUIVALENCE (E,D)
.
.
.
DO 10 I = M,N
E(I+1) = 2.0
D(I) = 3.0
10 CONTINUE
```

If E and D are dimensioned to 100 elements, no dependency is detected.

```
DIMENSION A(100),B(100)
EQUIVALENCE (A(50),B)
.
.
.
DO 15 I = 1,100
A(I) = X
B(I) = Y
15 CONTINUE
```

REFERENCE MADE TO AN ARRAY THAT IS NOT SUBSCRIPTED

```
DO 20 I = 2,N
A(I) = SASUM(N-I,A,1)
20 CONTINUE
```

ARRAY USED AS AN ARGUMENT TO A SUBROUTINE/FUNCTION

```
DO 40 I = M,N
A(I) = 2.0
CALL SB(A(I))
40 CONTINUE
```

DEFINITION AND REFERENCE HAVE A DIFFERENT NUMBER OF SUBSCRIPTS

```
DO 50 I = M,50
A(I,K) = A(I)
50 CONTINUE
```

```
DO 30 I = 1,100
B(I) = A(2,I)
A(I) = 2.0
30 CONTINUE
```

AMBIGUOUS OR CONFLICTING SUBSCRIPTS

```
DO 55 I = 1,100
B(I) = A(3)
A(I) = 3.0
55 CONTINUE
```

```
DO 60 I = 1,100
B(I) = A(I+N)
A(I) = 3.0
60 CONTINUE
```

```
DO 70 I = 50,1,-1
B(I) = A(I,I)
A(I,3) = 2.0
70 CONTINUE
```

```
DO 80 I = 1,N,2
B(I) = A(3*I-1)
A(2*I) = 2.0
80 CONTINUE
```

NULL DEPENDENCY WITH CII MODIFY BETWEEN DEFINITION AND REFERENCE

```
DO 10 I = 1,N
B(I) = A(J)
J = J+1
A(J) = 1.0
10 CONTINUE
```

```
DO 5 I = 1,N
A(J) = 3.0
J = J-2
A(J) = B(I)
5 CONTINUE
```

Null means the difference between the subscripts is zero. If the subscript difference is not equal to zero, vectorization is possible.

AMBIGUOUS INCREMENT OF CII

```
DO 20 I = M,N,K  
A(I) = A(I+1)  
20 CONTINUE
```

```
DO 30 I = 1,100  
A(J) = 1.0  
B(J) = A(J-1)  
J = J-K  
30 CONTINUE
```

DEPENDENCY POSSIBLE WITH ZERO INCREMENT

```
DO 90 I = 1,M  
A(J) = A(J)+B(I)  
J = J+N  
90 CONTINUE
```

This message is only issued if OPT=ZEROINC is specified on the CFT control statement.

NO CII WAS FOUND IN ARRAY REFERENCE

```
DO 55 I = 1,N  
A(I) = 1.0  
B(J) = B(K)  
55 CONTINUE
```


INDEX

- A edit descriptor, (2)6-21
- A (alphanumeric) editing, (2)6-21
- Access
 - direct
 - dataset position, (2)5-5
 - operations, (2)5-5
 - random
 - dataset creation for, E-13
 - modifying a record under, E-15
 - positioning while connected for, E-13
 - sequential
 - dataset position, (2)5-5
 - operations, (2)5-4
- Active DO-loop, (2)4-11
- Actual
 - arguments
 - association, (1)4-10
 - character substring, (1)4-11
 - description, (1)4-9, 4-10
 - external functions, (1)4-5
 - subroutine reference, (1)4-2
 - array
 - declarator, (1)2-11, 2-15
 - description, (1)2-15
- Address field, (3)1-10
- Adjustable
 - array
 - declarator, (1)2-11
 - description, (1)2-16
 - dimensions, (1)2-16
- Aids, vectorization, B-6
- ALIGN directive
 - description, (3)1-28
 - format, (3)1-28
- Alphanumeric
 - character, (1)1-3
 - editing, (2)6-21
- Alternate return, (2)7-7
- American National Standards Institute, (ANSI), (1)1-1
- ANSI
 - character set, A-1
 - FORTRAN
 - character set, A-1
 - conformance to standard, (1)1-1
 - standards, deviations from, (1)1-2
 - X3.9-1978, (1)1-1
- Apostrophe and quotation mark editing, (2)6-7
- ARGPLIMQ flow trace routine option, (3)1-24
- Argument
 - actual
 - description, (1)4-9, 4-10
- Argument (continued)
 - external functions, (1)4-5
 - in a subroutine reference, (1)4-2
 - is a character substring, (1)4-11
 - association of dummy and actual arguments, (1)4-10
 - description, (1)4-9
 - dummy
 - array, (1)4-12
 - description, (1)4-9
 - procedures, (1)4-12
 - statement function, (2)7-3
 - type character, (1)4-11
 - undefined, (1)4-11
 - variable, (1)4-11
 - retrieval, F-1
 - transmission, F-1
- Arithmetic
 - assignment statement
 - execution, (2)3-1
 - format, (2)3-1
 - constant
 - description, (1)2-3
 - expression, (1)3-1
 - expression
 - data type, (1)3-6
 - description, (1)3-1
 - evaluation, (1)3-10
 - forms, (1)3-5
 - functions, general, B-2
 - IF statement
 - execution, (2)4-4
 - format, (2)4-4
 - two-branch, format, E-6
 - operands, (1)3-3
 - operators
 - description, (1)3-2
 - interpretation in expressions, (1)3-2
 - precedence, (1)3-3
 - relational expression, (1)3-12
- Arrangement and reference, array element (1)2-14
- Array
 - actual, (1)2-15
 - adjustable, (1)2-16
 - bounds checking directive (BOUNDS), (3)1-25
 - declarator
 - actual, (1)2-11, 2-15
 - adjustable, (1)2-11
 - assumed-size, (1)2-11
 - description, (1)2-10
 - dummy, (1)2-11, 2-15

Array (continued)

- format, (1)2-10
- types, (1)2-11
- description, (1)2-10
- dimensions, (1)2-10
- dummy, (1)2-15
- dummy argument, (1)4-12
- element
 - arrangement and reference, (1)2-14
 - data type, (1)2-2
 - defined, (1)2-21
 - description, (1)2-10
 - initially defined, (1)2-21
 - invariant, description, (3)2-2
 - invariant, used in a reduction array operation, (3)2-3
 - undefined, (1)2-21
- element name
 - description, (1)2-10, 2-12
 - format, (1)2-12
 - in an EQUIVALENCE statement, (2)2-12
- element order, (1)2-13
- name
 - description, (1)2-10
 - in an EQUIVALENCE statement, (2)2-12
 - use of, (1)2-17
- noncharacter, for format specification, E-15
- reference, vector, (3)2-3
- size, (1)2-12
- storage sequence, (1)2-12.1, 2-13
- symbolic name, (1)2-26
- vectorization with, (3)2-10

ASCII

- character set, A-1
- internal code, (1)1-3

ASSIGN statement

- execution, (2)3-3
- format, (2)3-3

Assigned GO TO statement

- execution, (2)4-3
- format, (2)4-3

Assignment statements

- arithmetic
 - execution, (2)3-1
 - format, (2)3-1
- ASSIGN, (2)3-3
- character
 - execution, (2)3-2
 - format, (2)3-2
- description, (2)3-1
- formatted data, E-7
- logical
 - execution, (2)3-2
 - format, (2)3-2
- type conversion, (1)3-8

Associated

- entities
 - partially, (1)2-19
 - totally, (1)2-19
- storage sequences, (1)2-19

Association

- actual arguments, (1)4-10
- common, (2)2-15
- description, (1)2-19

Association (continued)

- dummy arguments, (1)4-10
- entities
 - description, (1)2-19
 - restrictions, (1)4-13
- entry, in function subprograms, (2)7-9
- equivalence, (2)2-12
- statements, (2)2-11

Assumed-size array declarator, (1)2-11

- A, S, V, VL, and VM registers, F-1

Auxiliary character set, (1)1-4, A-1

BACKSPACE statement

- description, (2)5-17
- format, (2)5-16

B and T register use, F-1

Basic real constant, (1)2-5

BL directive, (3)1-24.1

Blank common block, (2)2-15

BLOCK BEGINS messages, (3)1-9

Block

- common
 - blank, (2)2-15
 - description, (1)4-14
 - directive, dynamic (DYNAMIC), (3)1-25
 - name, (1)2-25
 - named, (1)4-1, (2)2-15
 - size, (2)2-14
 - storage sequence, (2)2-14
 - symbolic name, (1)2-25
- data
 - subprogram, (1)2-26, 4-1, (2)7-11
 - symbolic name, subprogram, (1)2-26
- design, entry, F-1
- field, (3)1-11
- names and lengths in octal, table of, (3)1-11
- statement, conditional
 - description, (2)4-5
 - execution, (2)4-8

BLOCK DATA statement, (1)4-1, (2)7-11

BLOCK directive, (3)1-27

Block IF statement

- execution, (2)4-6
- format, (2)4-6

BN and BZ

- edit descriptors, (2)6-13
- editing, (2)6-13

Boolean (octal) constant, (1)2-8

Boolean (masking) expression, (1)3-17

Boolean

- constant, (1)2-8
- data, (1)2-8
- expression, (1)3-6
- functions, B-5
- type conversion, (1)3-10

Bounds, array

- checking directive (BOUNDS), (3)1-25

BOUNDS

- directive, (3)1-25
- options, (3)1-25

BUFFER IN statement (CFT extension)

- description, (2)5-29
- format, (2)5-31

BUFFER OUT statement (CFT extension)
 description, (2)5-29
 format, (2)5-31

CALL ENDRPV format, I-2
 CALL SETRPV format, I-1
 CALL statement
 execution, (2)7-5
 format, (2)7-5

Categories, entity, (3)2-2

CFT
 compiler, (1)1-1, (3)1-1, 2-1, D-1
 control statement, format, (3)1-1
 extensions
 BUFFER IN statement, (2)5-29
 BUFFER OUT statement, (2)5-29
 dollar sign editing, (2)6-10.1
 hexadecimal (Z) editing, (2)6-20
 IMPLICIT NONE statement, (2)2-10
 LENGTH function, (2)5-34
 NAMELIST statement, (2)5-23
 O (octal) editing, (2)6-19
 octal (O) editing, (2)6-19
 POINTER statement, (2)2-3
 R (right-justified) editing, (2)6-23
 UNIT function, (2)5-33
 Z (hexadecimal) editing, (2)6-20
 input, (3)1-5
 language
 character set, (1)1-3
 definition, (1)1-1
 elements, (1)1-3
 messages, D-1
 output, (3)1-8
 use of optimized routines by, (3)2-12
 Z option, G-1

Character
 alphanumeric, (1)1-3
 assignment statement
 execution, (2)3-2
 format, (2)3-2
 constant, (1)2-8
 expression evaluation, (1)3-11
 expressions, (1)3-11
 functions, B-4
 primary, (1)3-11
 print control, (2)5-15
 relational expression, (1)3-13
 set
 ANSI FORTRAN, A-1
 ASCII, A-1
 auxiliary, (1)1-4, A-1
 Cray FORTRAN, A-1
 description, (1)1-3
 FORTRAN, (1)1-3, A-1
 special, (1)1-4
 substring
 actual argument, (1)4-11
 description, (1)2-18
 type, dummy argument, (1)4-11

CHARACTER type statement
 description, (2)2-8
 format, (2)2-8

Checking, bounds
 array, directive (BOUNDS), (3)1-25

CII (constant increment integer), (3)2-2
cilist (control information list),
 format, (2)5-9

Classes of symbolic names, (1)2-25

CLEARFI subroutine, (3)2-15

CLOSE
 specifiers and their meanings, (2)5-23
 statement
 execution, (2)5-21
 format, (2)5-21

Code
 internal, ASCII, (1)1-3
 source, (3)1-5

CODE directive
 description, (3)1-15
 format, (3)1-15

Collating sequence, (1)1-3

Colon
 description, (2)6-10
 editing, (2)6-10

Comment line, (1)1-9

Common
 association, (2)2-15
 block
 blank, (2)2-15
 description, (1)2-25, 4-14
 directive, dynamic (DYNAMIC),
 (3)1-25
 name, (1)2-25
 named, (1)4-1, (2)2-15
 size, (2)2-14
 storage sequence, (2)2-14
 symbolic name, (1)2-25

COMMON statement
 blank common, (2)2-15
 common association, (2)2-15
 description, (2)2-13
 format, (2)2-13
 restrictions, (2)2-15
 size, (2)2-14
 storage sequence, (2)2-14

Compiler
 definition, (1)1-1, (3)1-1, 2-1, D-1
 directive line, (1)1-10, (3)1-13
 directives, (3)1-12.2, 1-13
 options, (3)1-6, 1-12

Compile-time messages, D-2

Complex
 constant
 description, (1)2-7
 range, nonzero, (1)2-7
 data, (1)2-6, (2)6-19
 editing, (2)6-19
 expression, (1)3-6
 type conversion, (1)3-10

COMPLEX type statement, (2)2-7

Computed GO TO statement
 execution, (2)4-2
 format, (2)4-2

Computer system
 CRAY-1, (1)1-1
 CRAY X-MP, (1)1-1

Conditional
 block statement
 description, (4)4-5

Conditional (continued)
 execution, (2)4-8
 replacement statement, (3)2-13
 statements, (3)2-13
 Conformance with the ANSI standard, (1)1-1
 Connection, dataset, E-13
 Constant
 arithmetic, (1)2-3
 basic real, (1)2-5
 Boolean (octal), (1)2-8
 character, (1)2-8
 complex, (1)2-7
 description, (1)1-5, 2-2
 double-precision, (1)2-6
 expression, arithmetic, (1)3-1
 followed by a
 double-precision exponent, (1)2-6
 real exponent, (1)2-5
 Hollerith, E-2
 increment integer (CII), (3)2-2
 integer, (1)2-3, 2-4
 logical
 description, (1)2-7
 representation, (1)2-7
 nonzero
 complex, (1)2-7
 double-precision, (1)2-6
 real, (1)2-5
 optionally signed, (1)2-3
 range
 nonzero complex, (1)2-7
 nonzero double-precision, (1)2-6
 nonzero real, (1)2-5
 real, (1)2-4
 signed, (1)2-3
 symbolic name, (1)2-27
 unsigned, (1)2-3
 Continuation line, (1)1-9
 CONTINUE statement
 execution, (2)4-14
 format, (2)4-14
 Control
 character, print, (2)5-15
 directives
 integer (INT24, INT64), (3)1-20.1
 listable output, (3)1-14
 vectorization, (3)1-16
 format, positioning, (2)6-7
 information list (*cilist*), format
 (2)5-9
 processing, loop, (2)4-13
 statement
 CFT, format, (3)1-1
 program, (2)4-1
 subroutines, user, (2)5-26, 5-28
 Conventions, (1)1-2
 Conversion
 integer length, B-4
 type
 Boolean, (1)3-10
 complex, (1)3-10
 description, (1)3-9
 double-precision, (1)3-9
 functions, B-4
 in assignment statements, (1)3-8

Conversion (continued)
 integer, (1)3-9
 real, (1)3-9
 uppercase/lowercase, (1)1-5
 COS (Cray Operating System), (1)1-1
 Count, iteration, (2)4-12
 CRAY-1 Computer System, (1)1-1
 CRAY X-MP Computer System, (1)1-1
 Cray FORTRAN (CFT)
 character set, A-1
 compiler, (1)1-1, (3)1-1, 2-1
 intrinsic functions, B-1
 language, (1)1-1
 programming, (3)2-1
 utility procedures, C-1
 Cray Operating System (COS), (1)1-1
 Creating
 a dataset for random access, E-13
 a file, (2)5-3
 non-FORTRAN procedures, F-1
 Cross reference information, (3)1-12.1

 D editing, (2)6-17
 Data
 assignment, formatted, E-7
 block, subprogram, (1)2-26, 4-1, (2)7-11
 Boolean, (1)2-8
 complex, (1)2-6, (2)6-19
 description, (1)2-1
 double-precision, (1)2-5
 editing, (2)6-19
 integer, (1)2-3
 item, integer, (1)2-3
 length
 type statement, format, E-11
 with data types, E-12
 logical, (1)2-7
 real, (1)2-4
 specification, (2)2-1
 association, (2)2-1
 declaration, (2)2-1
 initialization, (2)2-1
 statements, (2)2-1
 type, (2)2-1, 2-7
 transfer, (2)5-11
 dataset position after, (2)5-15
 dataset position before, (2)5-5
 description, (2)5-12, 5-13
 direction of, (2)5-12
 execution, (2)5-12
 formatted, (2)5-14
 unformatted, (2)5-14
 type, (1)2-1
 arithmetic expressions, (1)3-6
 array element, (1)2-2
 edit descriptors with, (2)6-5
 function, (1)2-2
 function subprogram, (1)2-2
 DATA statement
 definition, (2)2-4
 features, E-10
 format, (2)2-5
 implied-DO list in a, (2)2-5
 restrictions, (2)2-6

Dataset
 connection, E-13
 creation for random access, E-13
 description, (2)5-3
 identifier, format, (2)5-7
 position
 after data transfer, (2)5-15
 before data transfer, (2)5-5
 unblocked, H-1
 Date functions, B-5
 DEBUG directive, (3)1-30
 Debug package, symbolic, G-1
 Debugging directives (DEBUG, NODEBUG),
 (3)1-30
 Declaration statements
 description, (2)2-1
 double
 description, E-10
 FUNCTION, format, E-10
 type, format, E-10
 Declarator
 array
 actual, (1)2-11, 2-15
 adjustable, (1)2-11
 assumed-size, (1)2-11
 description, (1)2-10
 dummy, (1)2-11, 2-15
 format, (1)2-10
 types, (1)2-11
 dimension
 description, (1)2-10
 lower and upper bounds, (1)2-11
 DECODE statement
 description, E-9
 format, E-7
 Defined
 entities, (1)2-21
 initially, variable or array element,
 (1)2-21
 variable or array element, (1)2-21
 Deleting a file, (2)5-3
 Dependencies, (3)2-4
 Dependency messages, informative, D-21
 Descriptor
 edit
 A, (2)6-21
 BN and BZ, (2)6-13
 description, (2)6-3, E-9
 dollar sign, (2)6-10.1
 E, (2)6-16
 F, (2)6-14
 G, (2)6-17
 H, (2)6-8
 I, (2)6-13
 L, (2)6-20
 O (octal) (CFT extension), (2)6-19
 obsolete, format, E-9
 P, (2)6-11
 R, (2)6-23
 S, SP, and SS, (2)6-13
 slash, (2)6-10
 T, TL, TR, (2)6-8
 with data types, (2)6-5
 X, (2)6-8

Descriptor (continued)
 Z (hexadecimal) (CFT extension),
 (2)6-20
 nonrepeatable edit, format, (2)6-4
 repeatable edit, format, (2)6-3
 Design, entry block, F-1
 Designator, substring
 description, (1)2-18
 format, (1)2-18
 Deviations from the ANSI standards, (1)1-2
 Differences, named common and blank common,
 (2)2-15
 Dimension
 adjustable, (1)2-16
 array, (1)2-10
 declarator
 description, (1)2-10
 lower and upper bounds, (1)2-11
 DIMENSION statement
 description, (2)2-2
 format, (2)2-2
 Direct access
 dataset position, (2)5-5
 operations, (2)5-5
 Direction of data transfer, (2)5-12
 Directive
 ALIGN
 description, (3)1-28
 format, (3)1-28
 array bounds checking (BOUNDS), (3)1-25
 BL, (3)1-24.1
 BLOCK, (3)1-27
 BOUNDS, (3)1-25
 CODE
 description, (3)1-15
 format, (3)1-15
 compiler
 description, (3)1-12.2, 1-13
 line, (3)1-13
 control
 integer (INT24, INT64), (3)1-20.1
 listable output, (3)1-14
 vectorization, (3)1-16
 DEBUG, (3)1-30
 debugging, (3)1-30
 DYNAMIC
 description, (3)1-25
 format, (3)1-25
 dynamic common block (DYNAMIC)
 description, (3)1-25
 format, (3)1-25
 EJECT
 description, (3)1-14
 format, (3)1-14
 FASTMD, (3)1-21
 FLOW, (3)1-21
 flow trace (FLOW/NOFLOW), (3)1-21
 INT24
 description, (3)1-20.1
 format, (3)1-21
 INT64
 description, (3)1-20.1
 format, (3)1-21
 integer control (INT24, INT64)
 description, (3)1-20.1
 format, (3)1-21

Directive (continued)

IVDEP
 description, (3)1-19
 format, (3)1-19
 line, compiler, (1)1-10, (3)1-13
 LIST
 description, (3)1-14
 format, (3)1-15
 listable output control, (3)1-14
 multiply/divide (FASTMD, SLOWMD),
 (3)1-21
 NEXTSCALAR
 description, (3)1-20
 format, (3)1-20
 NOBL, (3)1-24.1
 NOCODE
 description, (3)1-15
 format, (3)1-15
 NODEBUG, (3)1-30
 NODOREP
 description, (3)1-29
 format, (3)1-29
 NOFLOW, (3)1-21
 NOIFCON
 description, (3)1-28
 format, (3)1-28
 NOLIST
 description, (3)1-15
 format, (3)1-15
 NORECURRENCE
 description, (3)1-18
 format, (3)1-18
 NO SIDE EFFECTS
 description, (3)1-27
 format, (3)1-27
 NOVECTOR
 description, (3)1-17
 format, (3)1-17
 optimization, (3)1-26
 RESUMEDOREP
 description, (3)1-29
 format, (3)1-29
 RESUMEIFCON
 description, (3)1-29
 format, (3)1-29
 SAFEIF, (3)1-24.1
 scheduler, (3)1-24
 SHORTLOOP
 description, (3)1-20
 format, (3)1-20
 SLOWMD, (3)1-21
 UNSAFEIF, (3)1-24.1
 VECTOR
 description, (3)1-16
 format, (3)1-16
 vectorization control, (3)1-16
 VFUNCTION
 description, (3)1-19
 format, (3)1-19
 Disable, flow trace, (3)1-22
 Disjunct, logical
 form, (1)3-16
 value, (1)3-16
 Dollar sign
 edit descriptor, (2)6-10.1
 editing, (2)6-10.1

DO-loop
 active, (2)4-11
 description, (2)4-8
 execution of the range, (2)4-13
 extended range, E-15
 inactive, (2)4-11
 range, (2)4-10
 transfer into range, (2)4-14
 vectorizable, (3)2-1
 DO statement
 execution, (2)4-11
 format, (2)4-8
 incrementation processing, (2)4-13
 loop control processing, (2)4-13
 range, (2)4-10
 terminal statement
 description, (2)4-10
 execution, (2)4-13
 DO variable, (2)4-10
 Double declaration
 FUNCTION statement, format, E-10
 statement, E-10
 type statement, format, E-10
 Double-precision
 constant
 description, (1)2-6
 range, nonzero, (1)2-6
 data, (1)2-5
 editing, (2)6-17
 exponent, (1)2-6
 expression, (1)3-6
 type conversion, (1)3-9
 DOUBLE PRECISION type statement, (2)2-7
 Dummy
 arguments
 arrays, (1)4-12
 association, (1)4-10
 description, (1)4-9
 procedures, (1)4-12
 statement function, (2)7-3
 type character, (1)4-11
 undefined, (1)4-11
 variables, (1)4-11
 array
 declarator, (1)2-11, 2-15
 description, (1)2-15
 procedure, symbolic name, (1)2-28
 Dynamic common block directive (DYNAMIC)
 description, (3)1-25
 format, (3)1-25
 DYNAMIC directive
 description, (3)1-25
 format, (3)1-25
 E
 edit descriptors, (2)6-16
 editing, (2)6-16
 Edit descriptor
 A, (2)6-21
 BN and BZ, (2)6-13
 description, (2)6-3, E-9
 dollar sign, (2)6-10.1
 E, (2)6-16

Edit descriptor (continued)

- F, (2)6-14
- G, (2)6-17
- H, (2)6-8
- hexadecimal (Z) (CFT extension), (2)6-20
- I, (2)6-13
- L (logical), (2)6-20
- nonrepeatable, formats, (2)6-4
- O (octal) (CFT extension), (2)6-19
- obsolete, format, E-9
- P, (2)6-11
- R, (2)6-23
- repeatable, formats, (2)6-3
- S, SP, and SS, (2)6-13
- slash, (2)6-10
- T, TL, and TR, (2)6-8
- with data types, (2)6-5
- X, (2)6-8
- Z (hexadecimal) (CFT extension), (2)6-20

Editing

- A (alphanumeric), (2)6-21
- apostrophe and quotation mark, (2)6-7
- BN and BZ, (2)6-13
- colon, (2)6-10
- complex, (2)6-19
- D, (2)6-17
- data, (2)6-19
- description, (2)5-1
- dollar sign, (2)6-10.1
- E, (2)6-16
- F, (2)6-14
- G, (2)6-17
- H, (2)6-8
- hexadecimal (Z) (CFT extension), (2)6-20
- I, (2)6-13
- integer, (2)6-13
- L (logical), (2)6-20
- numeric (BN, BZ, S, SP, SS, I, F, E, D, and G), (2)6-12
- O (octal) (CFT extension), (2)6-19
- P, (2)6-11
- positional (T, TL, TR, and X), (2)6-8
- R (right-justified) (CFT extension), (2)6-23
- S, SP, and SS, (2)6-13
- slash, (2)6-10
- T, TL, and TR, (2)6-9
- X, (2)6-9
- Z (hexadecimal) (CFT extension), (2)6-20

Effect of ALLOC, SAVEALL, and BTREG on variable allocation, (3)1-4.4

Efficiency, vectorization, (3)2-11

EJECT directive, format, (3)1-14

Element

- array
 - arrangement and reference, (1)2-14
 - data type, (1)2-2
 - defined, (1)2-21
 - definition, (1)2-10
 - initially defined, (1)2-21
 - invariant, description, (3)2-2
 - invariant, used in a reduction array operation, (3)2-3
 - undefined, (1)2-21
- name, array, (1)2-10, 2-12
- order, array, (1)2-13

Elements of the CFT language, (1)1-3

- ELSE-block, (2)4-7
- ELSE IF-block, (2)4-6
- ELSE IF statement
 - execution, (2)4-7
 - format, (2)4-7
- ELSE statement, (2)4-7
- Empty sequence, (1)1-5
- Enable/disable, flow trace, (3)1-22
- ENCODE statement
 - description, E-8
 - format, E-7
- Endfile record, (2)5-2
- ENDFILE statement
 - description, (2)5-17
 - format, (2)5-16
- END IF statement, (2)4-6
- End-of-data (EOD) record, (2)5-2
- End-of-file (endfile) record, (2)5-2
- END statement, (2)4-16
- Entities
 - associated
 - partially, (1)2-19
 - totally, (1)2-19
 - association of
 - description, (1)2-19
 - restrictions, (1)4-13
 - defined, (1)2-21
 - global, (1)2-24
 - local, (1)2-24
 - undefined, (1)2-22
- Entity categories, (3)2-2
- Entry
 - association in function subprograms, (2)7-9
 - block design, F-1
 - procedure subprogram, referencing, (2)7-9
- ENTRY statement
 - description, (2)7-8
 - format, (2)7-8
 - restrictions, (2)7-9
- EOD, (2)5-2
- EOF, IOEF, and IOSTAT functions, E-16
- Equivalence, mathematical, (1)3-10
- EQUIVALENCE statement
 - array
 - element names, (2)2-12
 - names, (2)2-12
 - association, (2)2-12
 - description, (2)2-11
 - format, (2)2-11
 - restrictions, (2)2-12, 2-15
- Error
 - messages
 - description, D-1
 - during program execution, (3)1-5
 - recovery, I/O, (2)5-34
 - traceback, F-1
- Establishing a format, (2)5-13
- Evaluation
 - arithmetic expressions, (1)3-10
 - character expression, (1)3-11
 - expressions, (1)3-19
 - functions, order, (1)3-19
 - subscript, (1)2-16

Executable
 program, (1)1-10, 1-11
 statement, (1)1-8

Execution
 arithmetic assignment statement, (2)3-1
 arithmetic IF statement, (2)4-4
 assigned GO TO statement, (2)4-3
 ASSIGN statement, (2)3-3
 block IF statement, (2)4-6
 CALL statement, (2)7-5
 character assignment statement, (2)3-2
 CLOSE statement, (2)5-21
 computed GO TO statement, (2)4-2
 conditional block statement, (2)4-8
 CONTINUE statement, (2)4-14
 data transfer I/O statement, (2)5-12
 DO-loop statement range, (2)4-13
 DO statement, (2)4-11
 ELSE IF statement, (2)4-7
 external function references, (1)4-5
 function references, (1)4-8
 INQUIRE by dataset name statement,
 (2)5-20
 INQUIRE by unit statement, (2)5-20
 logical
 assignment statement, (2)3-2
 IF statement, (2)4-5
 OPEN statement, (2)5-20
 program
 description, (1)1-12
 error messages, (3)1-5
 RETURN statement, (2)7-7
 sequence, normal, (1)1-12
 statement function reference, (1)4-4
 terminal statement, (2)4-13
 unconditional GO TO statement, (2)4-2

Exponent
 double-precision, (1)2-6
 real, (1)2-5

Exponential functions, B-3

Expressions
 arithmetic
 data type, (1)3-6
 description, (1)3-1
 evaluation, (1)3-10
 forms, (1)3-5
 relational, (1)3-12
 arithmetic constant, (1)3-1
 Boolean, (1)3-6
 Boolean (masking), (1)3-17
 character
 description, (1)3-11
 evaluation, (1)3-11
 relational, (1)3-13
 complex, (1)3-6
 description, (1)3-1
 double-precision, (1)3-6
 evaluation, (1)3-19
 Hollerith, E-4
 Hollerith relational
 description, E-5
 format, E-5
 integer, (1)3-6
 interpretation of arithmetic operators,
 (1)3-2

Expressions (continued)
 invariant, (3)2-2
 logical
 description, (1)3-14
 form, (1)3-14, 3-16
 interpretation, (1)3-15
 value, (1)3-16
 parenthesized, (1)3-20
 real, (1)3-6
 relational, (1)3-12
 types, (1)3-1
 Extended range of a DO-loop, E-15
 External
 function, (1)4-5
 actual arguments, (1)4-5
 execution, (1)4-5
 references, (1)4-5
 symbolic name, (1)2-25
 names, table, (3)1-12
 routines, (3)1-30
 unit
 identifier, (2)5-6
 specifier, format (2)5-18
 EXTERNAL statement, format, (2)7-10

F
 edit descriptor, (2)6-14
 editing, (2)6-14

Factor
 description, (1)3-4
 logical
 form, (1)3-15
 value, (1)3-16
 scale, (2)6-11

FASTMD directive, (3)1-21

Features
 DATA statement, E-10
 outmoded, E-1

Field
 address, (3)1-10
 block, (3)1-11
 definition, (2)6-7
 main usage, (3)1-10
 name, (3)1-10
 type, (3)1-10
 width, (2)6-7

File
 creating, (2)5-3
 deleting, (2)5-3
 description, (2)5-2
 identifier, internal, (2)5-6
 input/output, (2)5-2
 internal
 description, (2)5-4
 restrictions, (2)5-4
 positions, (2)5-3
 specifier, format, (2)5-18
 FLODUMP routine, (3)1-23
 FLOW directive, (3)1-21
 FLOWLIM flow trace routine option, (3)1-24
 FLOW/NOFLOW directives, (3)1-21
 Flow trace
 directives (FLOW/NOFLOW), (3)1-21
 enable/disable, (3)1-22
 routine options, (3)1-23

Form

- arithmetic expression, (1)3-5
- logical
 - disjunct, (1)3-16
 - expression, (1)3-14, 3-16
 - factor, (1)3-15
 - term, (1)3-15

Format

- ALIGN directive, (3)1-28
- arithmetic assignment statement, (2)3-1
- arithmetic IF statement, (2)4-4
- array
 - declarator, (1)2-10
 - element name, (1)2-12
- assigned GO TO statement, (2)4-3
- ASSIGN statement, (2)3-3
- BACKSPACE statement, (2)5-16
- BLOCK DATA statement, (2)7-11
- block IF statement, (2)4-6
- BUFFER IN (CFT extension), (2)5-31
- BUFFER OUT (CFT extension), (2)5-31
- CALL ENDRPV, I-2
- CALL SETRPV, I-1
- CALL statement, (2)7-5
- CFT control statement, (3)1-1
- character assignment statement, (2)3-2
- CHARACTER type statement, (2)2-8
- CLOSE statement, (2)5-21
- CODE directive, (3)1-15
- COMMON statement, (2)2-13
- computed GO TO statement, (2)4-2
- CONTINUE statement, (2)4-14
- control
 - information list (*cilist*), (2)5-9
 - positioning, (2)6-7
- dataset identifier, (2)5-7
- DATA statement, (2)2-5
- DECODE statement, E-7
- DIMENSION statement, (2)2-2
- DO statement, (2)4-8
- double declaration
 - FUNCTION statement, E-10
 - type statement, E-10
- Dynamic common block directive (*DYNAMIC*), (3)1-25
- DYNAMIC directive, (3)1-25
- EJECT directive, (3)1-14
- ELSE IF statement, (2)4-7
- ELSE statement, (2)4-7
- ENCODE statement, E-7
- END statement, (2)4-16
- ENDFILE statement, (2)5-16
- END IF statement, (2)4-6
- ENTRY statement, (2)7-8
- EQUIVALENCE statement, (2)2-11
- establishing a, (2)5-13
- EXTERNAL statement, (2)7-10
- external unit specifier, (2)5-18
- file specifier, (2)5-18
- format
 - identifier, (2)5-7
 - specification, (2)6-2
- FORMAT statement, (2)6-1
- function reference, (2)7-2
- FUNCTION statement, (2)7-2

Format (continued)

- Hollerith relational expression, E-5
- identifier, (2)5-7
- IMPLICIT NONE statement (CFT extension) (2)2-10
- IMPLICIT statement, (2)2-9
- implied-DO list, (2)5-11
- indirect logical IF statement, E-6
- input NAMELIST group record, (2)5-24
- INQUIRE by dataset name statement, (2)5-18
- INQUIRE by unit statement, (2)5-18
- INT24 directive, (3)1-21
- INT64 directive, (3)1-21
- integer control directives (INT24, INT64), (3)1-21
- INTRINSIC statement, (2)2-15
- I/O, list-directed, (2)6-24
- IVDEP directive, (3)1-19
- list-directed I/O, (2)6-24
- LIST directive, (3)1-15
- logical
 - assignment statement, (2)3-2
 - IF statement, (2)4-4
- NAMELIST statement, (2)5-23
- NEXTSCALAR directive, (3)1-20
- NOCODE directive, (3)1-15
- NODOREP directive, (3)1-29
- NOIFCON directive, (3)1-28
- NOLIST directive, (3)1-15
- nonrepeatable edit descriptors, (2)6-4
- NORECURRENCE directive, (3)1-18
- NO SIDE EFFECTS directive, (3)1-27
- NOVECTOR directive, (3)1-17
- obsolete edit descriptors, E-9
- OPEN statement, (2)5-20
- output NAMELIST group record, (2)5-28
- PARAMETER statement, (2)2-1
- PAUSE statement, (2)4-15
- POINTER statement, (2)2-3
- PRINT statement, (2)5-8
- PROGRAM statement, (2)7-1
- PUNCH statement, E-11
- READ statement, (2)5-8
- repeatable edit descriptors, (2)6-3
- reprieve
 - initiation, I-1
 - termination, I-2
- RESUMEDOREP directive, (3)1-29
- RESUMEIFCON directive, (3)1-29
- RETURN statement, (2)7-6, 7-7
- REWIND statement, (2)5-16
- SAVE statement, (2)2-16
- SHORTLOOP directive, (3)1-20
- specification
 - description, (2)6-1
 - format, (2)6-2
 - Hollerith, E-6
 - interaction with I/O lists, (2)6-5
 - noncharacter arrays, E-15
- statement function definition
 - statement, (2)7-3
- STOP statement, (2)4-15
- SUBROUTINE statement, (2)7-6
- substring designator, (1)2-18
- two-branch arithmetic IF statement, E-6

- Format (continued)
 - type statement, (2)2-7
 - type statements with data length, E-11
 - unconditional GO TO statement, (2)4-1
 - unit identifier, (2)5-6
 - VECTOR directive, (3)1-16
 - VFUNCTION directive, (3)1-19
 - WRITE statement, (2)5-8
- FORMAT statement, (2)6-1
- Formatted
 - data
 - assignment, E-7
 - transfer, (2)5-14
 - records, (2)5-1
- FORTRAN
 - character set, (1)1-3, A-1
 - compiler, (1)1-1, (3)1-1, 2-1, D-1
 - intrinsic functions, B-1
 - language
 - definition, (1)1-1
 - elements, (1)1-3
 - specifications, (1)1-1
 - syntactic items, (1)1-5
 - programming, (3)2-1
 - statement, description, (1)1-8, (2)1-1
 - utility procedures, C-1
- FORTRAN 77, (1)1-1
- Function
 - Boolean, B-5
 - character, B-4
 - data type, (1)2-2
 - definition statement format, statement, (2)7-3
 - dummy argument, statement, (2)7-3
 - EOF, E-16
 - execution of references, (1)4-7
 - exponential, B-3
 - external
 - actual arguments, (1)4-5
 - description, (1)4-5
 - execution, references, (1)4-5
 - referencing, (1)4-5
 - symbolic name, (1)2-25
 - general arithmetic, B-2
 - IEOF, E-16
 - intrinsic
 - description, (1)4-6, 2-2, B-1
 - names, (2)2-16, B-1
 - referencing, (1)4-6
 - restrictions, (1)4-6
 - symbolic name, (1)2-28
 - IOSTAT, E-16
 - LENGTH (CFT extension), (2)5-34
 - logarithmic, B-3
 - maximum/minimum, B-4
 - miscellaneous, B-6
 - name, intrinsic, (2)2-16
 - order of evaluation, (1)3-19
 - reference, format, (2)7-2
 - references, execution, (1)4-8
 - referencing, (1)4-4, 4-8
 - restrictions, (1)4-7
 - statement
 - description, (1)4-3
 - execution, (1)4-4
 - referencing, (1)4-4
 - restrictions, (1)4-4
 - symbolic name, (1)2-28
 - subprogram
 - data type, (1)2-2
 - description, (1)4-3, (2)7-2
 - entry association, (2)7-9
 - restrictions, (1)4-7
 - time and date, B-5
 - trigonometric, B-3
 - type conversion, B-4
 - UNIT (CFT extension), (2)5-33
 - utility
 - LENGTH (CFT extension), (2)5-34
 - UNIT (CFT extension), (2)5-33
 - values, F-1
- FUNCTION statement
 - double declaration, format, E-10
 - format, (2)7-2
- G
 - edit descriptors, (2)6-17
 - editing, (2)6-17
- General
 - arithmetic functions, B-2
 - guidelines for vectorization, (3)2-16
- Generic and specific intrinsic function names, B-7
- GETPOS/SETPOS, E-13
- Global entities, (1)2-24
- GO TO statement
 - assigned
 - execution, (2)4-3
 - format, (2)4-3
 - computed
 - execution, (2)4-2
 - format, (2)4-2
 - unconditional
 - execution, (2)4-2
 - format, (2)4-1
- Group
 - name, NAMELIST, (1)2-29
 - record
 - input NAMELIST, format, (2)5-24
 - output NAMELIST, format, (2)5-28
- Guidelines for vectorization, (3)2-16
- H
 - edit descriptor, (2)6-8
 - editing, (2)6-8
- Header lines, page, (3)1-8
- Hexadecimal (Z) editing, (2)6-20
- High-level language, (1)1-1
- Hollerith
 - constants, E-2
 - expression, E-4
 - format specification, E-6
 - relational expression
 - description, E-5
 - format, E-5

I

- edit descriptors, (2)6-13
- editing, (2)6-13
- Identifiers
 - dataset, format, (2)5-7
 - description, (2)5-6
 - format, format, (2)5-7
 - internal file, (2)5-6
 - unit
 - external, (2)5-6
 - format, (2)5-6
- Identifying a unit, (2)5-13
- IEOF function, E-16
- IF-block, (2)4-5
- IF-levels and blocks, (2)4-9
- IF statement
 - arithmetic
 - execution, (2)4-4
 - format, (2)4-4
 - block
 - execution, (2)4-6
 - format, (2)4-6
 - END, format, (2)4-6
 - indirect logical
 - description, E-6
 - format, E-6
 - logical
 - execution, (2)4-5
 - format, (2)4-4
 - two-branch arithmetic
 - description, E-6
 - format, E-6
- IMPLICIT NONE statement (CFT extension)
 - description, (2)2-10
 - format, (2)2-10
- IMPLICIT statement
 - description, (2)2-9
 - format, (2)2-9
- Implied-DO list
 - description, (2)5-11
 - format, (2)5-11
 - in a DATA statement, (2)2-5
 - range, (2)2-6, 5-12
- Inactive DO-loop, (2)4-11
- Incrementation
 - parameter, (2)4-11
 - processing, (2)4-13
- Increment integer, constant (CII), (3)2-2
- Indirect logical IF statement
 - description, E-6
 - format, E-6
- Information
 - cross reference, (3)1-12.1
 - list, control (*cilist*), (2)5-9
- Informative dependency messages, D-21
- Initial
 - line, (1)1-9
 - parameter, (2)4-11
- Initialization statements, (2)2-1
- Initially defined variable or array
 - element, (1)2-21
- Initiation, retrieve
 - description, I-1
 - format, I-1
- Input
 - list-directed, (2)6-24
 - list item, (2)5-10
 - NAMELIST
 - description, (2)5-24
 - group record, format, (2)5-24
 - physical record, (2)5-24
 - operations, random, E-13
 - processing, NAMELIST, (2)5-26
 - statement, (2)5-1
 - to CFT, (3)1-5
 - variables, NAMELIST, (2)5-25
- Input/output
 - error recovery, (2)5-34
 - file, (2)5-2
 - list (*iolist*), (2)5-10
 - list-directed, (2)6-24
 - operations, random, E-13
 - records, (2)5-1
 - statements
 - description, (2)5-1
 - restrictions, (2)5-34
- INQUIRE by dataset name statement
 - execution, (2)5-20
 - format, (2)5-18
- INQUIRE by unit statement
 - execution, (2)5-20
 - format, (2)5-18
- INQUIRE statement
 - description, (2)5-17
 - restrictions, (2)5-20
- Inquiry specifiers and their meanings, (2)5-19
- INT24 directive
 - description, (3)1-20.1
 - format, (3)1-21
- INT64 directive
 - description, (3)1-20.1
 - format, (3)1-21
- Integer
 - constant, (1)2-3, 2-4
 - constant increment (CII), (3)2-2
 - control directives (INT24, INT64)
 - description, (3)1-20.1
 - format, (3)1-21
 - data, (1)2-3
 - data item, (1)2-3
 - editing, (2)6-13
 - expression, (1)3-6
 - length conversion, B-4
 - quotient, (1)3-8
 - type conversion, (1)3-9
- INTEGER type statement, (2)2-7
- Interaction between I/O lists and format specification, (2)6-5
- Internal
 - code, ASCII, (1)1-3
 - file
 - description, (2)5-4
 - identifier, (2)5-6
 - restrictions, (2)5-4
 - records, (2)5-4
 - representation, (2)6-7

Interpretation
 arithmetic operators in expressions,
 (1)3-2
 logical expression, (1)3-14
 rules, (1)3-20
 Intrinsic function
 data type, (1)2-2
 definition, (1)4-6
 description, B-1
 examples, (1)4-6
 names, (2)2-16, B-7
 referencing, (1)4-6
 restrictions, (1)4-6
 symbolic name, (1)2-28
 INTRINSIC statement
 description, (2)2-15
 format, (2)2-15
 Invariant
 array element
 description, (3)2-2
 used in a reduction array operation,
 (3)2-3
 description, (3)2-2
 expression, (3)2-2
 I/O
 error recovery, (2)5-34
 list-directed
 description, (2)6-24
 format, (2)6-24
 lists, interaction with format
 specifications, (2)6-5
 iolist (input/output list), (2)5-10
 IOSTAT function, E-16
 Items
 data
 integer, (1)2-3
 list
 description, (1)1-7
 input, (2)5-10
 output, (2)5-11
 syntactic, (1)1-5, 1-7
 Iteration count, (2)4-12
 IVDEP directive
 description, (3)1-19
 format, (3)1-19

 Keyword, (1)1-7

 L
 edit descriptor, (2)6-20
 (logical) editing, (2)6-20
 Label, statement, (1)1-6
 Language
 CFT
 definition, (1)1-1
 elements, (1)1-3
 Cray FORTRAN, (1)1-1
 high-level, (1)1-1
 len (length specification), (2)2-8, 6-22

Length
 data
 type statement, E-11
 type statements with, format, E-11
 with data types, E-12
 specification (*len*), (2)2-8, 6-22
 word, octal, with block names, (3)1-11
 LENGTH function (CFT extension), (2)5-34
 LENGTH, utility function, (2)5-34
 Library routine SYMDEBUG, G-1
 Lines
 comment, (1)1-9
 compiler directive, (1)1-10, (3)1-13
 continuation, (1)1-9
 description, (1)1-8
 initial, (1)1-9
 order, (1)1-13
 page header, (3)1-8
 terminal, (1)1-9, 1-10
 List
 control information (*cilist*), (2)5-9
 description, (1)1-7
 implied-DO
 description, (2)5-11
 range, (2)2-6, 5-12
 input/output (*iolist*), (2)5-10
 I/O, interaction with format
 specifications, (2)6-5
 item
 description, (1)1-7
 input, (2)5-10
 output, (2)5-11
 Listable output
 control directives, (3)1-14
 description, (3)1-8
 List-directed
 input, (2)6-24
 I/O
 description, (2)6-24
 format, (2)6-24
 output, (2)6-26
 LIST directive
 description, (3)1-14
 format, (3)1-15
 Listings, source statement, (3)1-8.1
 Local
 entities, (1)2-24
 temporary variable reference, F-1
 Logarithmic functions, B-3
 Logfile messages, D-19
 Logical
 assignment statement
 execution, (2)3-2
 format, (2)3-2
 constant
 description, (1)2-7
 representation, (1)2-7
 data, (1)2-7
 disjunct
 form, (1)3-16
 value, (1)3-16
 editing (L), (2)6-20

- expression
 - description, (1)3-14
 - form, (1)3-16
 - form and interpretation, (1)3-14
 - value, (1)3-16
- factor
 - form, (1)3-15
 - value, (1)3-16
- IF statement
 - execution, (2)4-5
 - format, (2)4-4
 - indirect, format, E-6
 - interpretation, (1)3-15
 - operands, (1)3-15
 - operators, (1)3-14
 - primaries, (1)3-15
- term
 - form, (1)3-15
 - value, (1)3-16
- Logical IF statement, (2)4-4
- LOGICAL type statement, (2)2-7
- Loop control processing, (2)4-13
- Loops encountered, table, (3)1-12
- Lower and upper bounds of dimension
 - declarators, (1)2-11
- Lowercase conversion, (1)1-5
- Main
 - program
 - description, (1)1-12, (2)7-1
 - symbolic name, (1)2-26
 - usage field, (3)1-10
 - Masking expression, Boolean, (1)3-17
 - Mathematical equivalence, (1)3-10
 - Maximum/minimum functions, B-4
- Messages
 - BLOCK BEGINS, (3)1-9
 - CFT, D-1
 - compile-time, D-2
 - description, (3)1-12.1
 - error
 - description, D-1
 - during program execution, (3)1-5
 - informative dependency, D-21
 - logfile, D-19
- Miscellaneous functions, B-6
- Modes
 - scalar, (3)2-4
 - vector, (3)2-4
- Modifying a record under random access, E-15
- Multiply/divide directives (FASTMD, SLOWMD), (3)1-21
- Name
 - array
 - definition, (1)2-10
 - in an EQUIVALENCE statement, (2)2-12
 - use of, (1)2-17
 - array element
 - definition, (1)2-10, 2-12
 - in an EQUIVALENCE statement, (2)2-12
 - block, table, (3)1-11
 - common block, (1)2-25
 - external, table, (3)1-11
 - field, (3)1-9, 1-10
 - group, NAMELIST, (1)2-29
 - intrinsic function, (2)2-16, B-1, B-7
 - symbolic
 - array, (1)2-26
 - block data subprogram, (1)2-26
 - classes, (1)2-25
 - common block, (1)2-25
 - constant, (1)2-27
 - description, (1)2-23
 - dummy procedure, (1)2-28
 - external function, (1)2-25
 - intrinsic function, (1)2-28
 - main program name, (1)2-26
 - scope, (1)2-24
 - statement function, (1)2-28
 - subroutine, (1)2-26
 - variable, (1)2-27
 - variable, (1)2-27
- Named common blocks, (1)4-1, (2)2-15
- NAMELIST
 - group name, (1)2-29
 - input
 - description, (2)5-24
 - group record, format, (2)5-24
 - physical record, (2)5-24
 - processing, (2)5-26
 - variables, (2)5-25
 - output
 - description, (2)5-28
 - group record, format, (2)5-28
 - statement (CFT extension), format, (2)5-23
- Names, symbolic
 - classes, (1)2-26
 - description, (1)1-6, 2-25
 - scope, (1)2-25
- Names encountered, table, (3)1-9
- NEXTSCALAR directive
 - description, (3)1-20
 - format, (3)1-20
- NOBL directive, (3)1-24.1
- NOCODE directive
 - description, (3)1-15
 - format, (3)1-15
- NODEBUG directive, (3)1-30
- NODOREP directive
 - description, (3)1-29
 - format, (3)1-29
- NOFLOW directive, (3)1-21
- NOIFCON directive
 - description, (3)1-28
 - format, (3)1-28
- NOLIST directive
 - description, (3)1-15
 - format, (3)1-15
- Non-ANSI features, outmoded, E-1
- Noncharacter arrays for format
 - specification, E-15
- Non-executable statement, (1)1-8
- Non-FORTRAN
 - procedures, creating, F-1

- subprograms
 - creation of, F-1
 - description, (1)4-9
 - programming, F-1
- Nonrepeatable edit descriptors, formats, (2)6-4
- Nonzero
 - complex constant
 - description, (1)2-7
 - range, (1)2-7
 - double-precision constant
 - description, (1)2-6
 - range, (1)2-6
 - real constant
 - description, (1)2-5
 - range, (1)2-5
- NORECURRENCE directive
 - description, (3)1-18
 - format, (3)1-18
- Normal execution sequence, (1)1-12
- NO SIDE EFFECTS directive
 - description, (3)1-27
 - format, (3)1-27
- Notation
 - syntax of FORTRAN statement forms, (1)1-2
- NOVECTOR directive
 - description, (3)1-17
 - format, (3)1-17
- Numbers, table of statement, (3)1-9
- Numeric editing, (BN, BZ, S, SP, SS, I, F, E, D, and G), (2)6-12
- O (octal)
 - edit descriptor, (2)6-19
 - editing (CFT extension), (2)6-19
- Obsolete edit descriptors, format, E-9
- Octal
 - Boolean, constant, (1)2-8
 - (O) editing (CFT extension), (2)6-19
 - table of block names and lengths, (3)1-11
- OPEN
 - specifiers and their meanings, (2)5-22
 - statement
 - execution, (2)5-20
 - format, (2)5-20
- Operand
 - arithmetic, (1)3-3
 - description, (1)3-1
 - logical, (1)3-15
- Operations
 - direct access, (2)5-5
 - random input/output, E-13
 - sequential access, (2)5-4
- Operator
 - arithmetic
 - description, (1)3-1, 3-2
 - interpretation in expressions, (1)3-2
 - precedence, (1)3-3
 - description, (1)1-7
 - logical, (1)3-14
 - precedence, (1)3-18
 - relational, (1)3-12

- Optimization
 - directives, (3)1-26
 - options, (3)1-3
- Optimized routines, (3)2-11
- Optionally signed constant, (1)2-3
- Options
 - ARGPLIMQ, (3)1-24
 - BOUNDS, (3)1-25
 - CFT Z, G-1
 - compiler, (3)1-6, 1-12
 - FLOWLIM, (3)1-24
 - optimization, (3)1-3
 - routine, flow trace, (3)1-23
 - SETPLIMQ, (3)1-23
 - Z, G-1
- Order
 - element, array, (1)2-13
 - evaluation of functions, (1)3-19
 - lines, (1)1-13
 - statements, (1)1-13
- Outmoded features, E-1
- Output
 - control directives, listable, (3)1-14
 - from CFT, (3)1-8
 - listable, (3)1-8
 - list-directed, (2)6-26
 - list item, (2)5-11
 - NAMelist, format, (2)5-28
 - operations, random, E-13
 - statement, (2)5-1
- P
 - edit descriptor, (2)6-11
 - editing, (2)6-11
- Package, symbolic debug, G-1
- Page
 - header lines, (3)1-8
 - table, program unit, (3)1-12.1
- Parameter
 - incrementation, (2)4-11
 - initial, (2)4-11
 - terminal, (2)4-11
- PARAMETER statement
 - description, (2)2-1
 - format, (2)2-1
- Parentheses, (1)3-20
- Parenthesized expression, (1)3-20
- Partially associated entities, (1)2-19
- PAUSE statement, format, (2)4-15
- Physical record, input NAMelist, (2)5-24
- POINTER statement (CFT extension)
 - description, (2)2-3
 - format, (2)2-3
- Position
 - dataset
 - after data transfer, (2)5-15
 - before data transfer, (2)5-5
 - record and file, (2)5-3
- Positional editing (T, TL, TR, and X), (2)6-8
- Positioning
 - by format control, (2)6-7
 - while connected for random access (GETPOS/SETPOS), E-13

Precedence
 arithmetic operators, (1)3-3
 operators, (1)3-18
 Primary
 character, (1)3-11
 description, (1)3-4
 logical, (1)3-15
 Print control characters, (2)5-15
 Printing, (2)5-15
 PRINT statement, format, (2)5-8
 Procedures
 dummy argument, (1)4-12
 dummy, symbolic name, (1)2-28
 non-FORTRAN, creating, F-1
 utility, (1)4-6, C-1
 Procedure subprograms
 description, (1)4-2
 referencing, (2)7-9
 Processing
 incrementation, (2)4-13
 input, NAMELIST, (2)5-26
 loop control, (2)4-13
 reprieve, I-1
 Program
 control statements, (2)4-1
 executable, (1)1-10, 1-11
 execution
 description, (1)1-12
 error messages, (3)1-5
 main, (1)1-12, (2)7-1
 name, main, (1)2-26
 subprogram, (1)1-10, 1-12, (2)7-1
 unit
 description, (1)1-10, (2)7-1
 page table, (3)1-12.1
 specification, (2)7-1
 Programming, Cray FORTRAN, (3)2-1
 PROGRAM statement, format, (2)7-1
 Pseudo vector, (3)2-3
 PUNCH statement
 description, E-11
 format, E-11

 Qualifications for vectorization, (3)2-1
 Quotation mark editing, (2)6-7
 Quotient, integer, (1)3-8

 R
 edit descriptor, (2)6-23
 editing (right-justified) (CFT
 extension), (2)6-23
 Random
 access
 creating a dataset for, E-13
 modifying a record under, E-15
 positioning while connected for, E-13
 input/output operations, E-13
 Range
 DO-loop
 description, (2)4-10
 execution, (2)4-13
 extended, E-15
 transfer into, (2)4-14
 implied-DO list, (2)2-6, 5-12
 nonzero
 complex constant, (1)2-7
 double-precision constant, (1)2-6
 real constant, (1)2-5
 READ statement, format, (2)5-8
 Reading, (2)5-1
 READMS/WRITMS routines, E-14
 Real
 constant
 basic, (1)2-5
 description, (1)2-4
 nonzero, (1)2-5
 data, (1)2-4
 exponent, (1)2-5
 expression, (1)3-6
 type conversion, (1)3-9
 REAL type statement, (2)2-7
 Record
 description, (2)5-1
 end-of-data, (2)5-2
 end-of-file (endfile), (2)5-2
 formatted, (2)5-1
 group
 input NAMELIST, format, (2)5-24
 output NAMELIST, format, (2)5-28
 input/output, (2)5-1
 internal, (2)5-4
 modification under random access, E-15
 physical, input NAMELIST, (2)5-24
 positions, (2)5-3
 unformatted, (2)5-2
 Recovery, I/O error, (2)5-34
 Reference
 array element, (1)2-14
 cross, information, (3)1-12.1
 description, (1)2-1
 external function, execution, (1)4-5
 function
 execution, (1)4-8
 format, (2)7-2
 local temporary variable, F-1
 statement function, execution, (1)4-4
 subroutine
 actual arguments, (1)4-2
 description, (2)7-5
 vector array, (3)2-3
 Referencing
 external functions, (1)4-5
 functions, (1)4-8
 intrinsic functions, (1)4-6
 procedure subprogram entry, (2)7-9
 statement functions, (1)4-4
 Register use
 A, S, V, VL, and VM, F-1
 B and T, F-1
 Relational
 expressions
 arithmetic, (1)3-12
 character, (1)3-13
 description, (1)3-12
 Hollerith, format, E-5
 operators, (1)3-12
 Repeat specification, (2)6-3
 Repeatable edit descriptor, (2)6-3

Replacement statement, conditional, (3)2-13

Representation

internal, (2)6-7
logical constant, (1)2-7

Reprive

initiation
description, I-1
format, I-1
processing, I-1
termination
description, I-2
format, I-2

Required order of lines and statements,
(1)1-14

Restrictions

association of entities, (1)4-13
COMMON statement, (2)2-15
DATA statement, (2)2-6
ENTRY statement, (2)7-9
EQUIVALENCE statement, (2)2-12, 2-15
function subprogram, (1)4-7
input/output statements, (2)5-34
INQUIRE statement, (2)5-20
internal file, (2)5-4
intrinsic function, (1)4-6
statement functions, (1)4-4
subroutine subprogram, (1)4-3

RESUMEDOREP directive

description, (3)1-29
format, (3)1-29

RESUMEIFCON directive

description, (3)1-29
format, (3)1-29

Retrieval, argument, F-1

Return, alternate, (2)7-7

RETURN statement

description, (2)7-6
execution, (2)7-7
format, (2)7-6, 7-7

REWIND statement

description, (2)5-17
format, (2)5-16

Right-justified (R) editing (CFT
extension), (2)6-23

Routine

external, (3)1-30
FLODUMP, (3)1-23
flow trace, options, (3)1-23
library, SYMDEBUG, G-1
optimized, (3)2-11
READMS/WRITMS, E-14

Rules, interpretation, (1)3-20

SAFEIF directive, (3)1-24.1

SAVE statement

description, (2)2-16
format, (2)2-16

Scalar temporary, (3)2-3

Scale factor, (2)6-11

Scheduler directives, (3)1-24

Scope of symbolic name, (1)2-24

SENSEFI subroutine, (3)2-15

Sequence

collating, (1)1-3
definition, (1)1-5
empty, (1)1-5
normal execution, (1)1-12
storage
array, (1)2-12.1, 2-13
associated, (1)2-19
common block, (2)2-14
description, (1)2-19
size, (1)2-19

Sequential access

dataset positions, (2)5-5
operations, (2)5-4

SETFI subroutine, (3)2-15

SETPLIMQ flow trace routine option, (3)1-23

SETPOS, E-13

Sets, character

ANSI FORTRAN, A-1
ASCII, A-1
Cray FORTRAN, A-1
description, (1)1-3, A-1

SHORTLOOP directive

description, (3)1-20
format, (3)1-20

Signed constant, (1)2-3

Size

array, (1)2-12
common block, (2)2-14
storage sequence, (1)2-19

Slash

edit descriptor, (2)6-10
editing, (2)6-10

SLOWMD directive, (3)1-21

Source

code, (3)1-5
statement listing, (3)1-8.1

Space Tables

stack, (3)1-11
static, (3)1-11

Special characters, (1)1-4

Specification

data, (2)2-1
format
description, (2)6-1
format, (2)6-2
Hollerith, E-6
interaction with I/O lists, (2)6-5
noncharacter arrays for, E-15
length (*len*), (2)2-8
program unit, (2)7-1
repeat, (2)6-3
subprogram, (1)4-1

Specifier

CLOSE, and their meanings, (2)5-23
external unit, format, (2)5-18
file, format, (2)5-18
inquiry and meanings, (2)5-19
OPEN, and their meanings, (2)5-22

S, SP, and SS

edit descriptors, (2)6-13
editing, (2)6-13

Stack Space Table, (3)1-11

Standard, ANSI

conformance with, (1)1-1
deviations from, (1)1-2

Statement

arithmetic assignment
 execution, (2)3-1
 format, (2)3-1
 arithmetic IF
 execution, (2)4-4
 format, (2)4-4
 ASSIGN
 execution, (2)3-3
 format, (2)3-3
 assigned GO TO
 execution, (2)4-3
 format, (2)4-3
 assignment
 description, (2)3-1
 type conversion, (1)3-8
 association, (2)2-11
 BACKSPACE
 description, (2)5-17
 format, (2)5-16
 BLOCK DATA, format, (2)7-11
 block IF
 execution, (2)4-6
 format, (2)4-6
 BUFFER IN (CFT extension)
 description, (2)5-29
 format, (2)5-31
 BUFFER OUT (CFT extension)
 description, (2)5-29
 format, (2)5-31
 CALL
 execution, (2)7-5
 format, (2)7-5
 CFT control, format, (3)1-1
 character assignment
 execution, (2)3-2
 format, (2)3-2
 CLOSE
 execution, (2)5-21
 format, (2)5-21
 COMMON
 description, (2)2-13
 format, (2)2-13
 restrictions, (2)2-15
 computed GO TO
 execution, (2)4-2
 format, (2)4-2
 conditional, (3)2-13
 conditional block
 description, (2)4-5
 execution, (2)4-8
 conditional replacement, (3)2-13
 CONTINUE
 execution, (2)4-14
 format, (2)4-14
 DATA
 description, (2)2-4
 features, E-10
 format, (2)2-5
 restrictions, (2)2-6
 data specification, (2)2-1
 declaration, (2)2-1
 DECODE
 description, E-9
 format, E-7

Statement (continued)

DIMENSION
 description, (2)2-2
 format, (2)2-2
 DO
 execution, (2)4-11
 format, (2)4-8
 double declaration
 description, E-10
 FUNCTION, format, E-10
 type, format, E-10
 ELSE, (2)4-7
 ELSE-block, (2)4-7
 ELSE IF
 execution, (2)4-7
 format, (2)4-7
 ELSE IF-block, (2)4-6
 ENCODE
 description, E-8
 format, E-7
 END, format, (2)4-16
 ENDFILE
 description, (2)5-17
 format, (2)5-16
 END IF, format, (2)4-6
 ENTRY
 description, (2)7-8
 format, (2)7-8
 restrictions, (2)7-9
 EQUIVALENCE
 description, (2)2-11
 format, (2)2-11
 restrictions, (2)2-12, 2-15
 executable, (1)1-8
 EXTERNAL, format, (2)7-10
 FORMAT, format, (2)6-1
 FORTRAN, (1)1-8, (2)1-1
 FUNCTION, format, (2)7-2
 IF-block, (2)4-5
 IMPLICIT
 description, (2)2-9
 format, (2)2-9
 IMPLICIT NONE (CFT extension)
 description, (2)2-10
 format, (2)2-10
 indirect logical IF
 description, E-6
 format, E-6
 initialization, (2)2-1
 input/output
 description, (2)5-1
 restrictions, (2)5-34
 INQUIRE
 description, (2)5-17
 restrictions, (2)5-20
 INQUIRE by dataset name
 execution, (2)5-20
 format, (2)5-18
 INQUIRE by unit
 execution, (2)5-20
 format, (2)5-18
 INTRINSIC
 description, (2)2-15
 format, (2)2-15
 label, (1)1-6

Statement (continued)

- logical assignment
 - execution, (2)3-2
 - format, (2)3-2
- logical IF
 - execution, (2)4-5
 - format, (2)4-4
- NAMelist (CFT extension), format, (2)5-23
- non-executable, (1)1-8
- OPEN
 - execution, (2)5-20
 - format, (2)5-20
- order, (1)1-13
- PARAMETER
 - description, (2)2-1
 - format, (2)2-1
- PAUSE, format, (2)4-15
- POINTER (CFT extension)
 - description, (2)2-2
 - format, (2)2-2
- PRINT, format, (2)5-8
- PROGRAM, format, (2)7-1
- program control, (2)4-1
- PUNCH
 - description, E-11
 - format, E-11
- READ, format, (2)5-8
- RETURN
 - description, (2)7-6
 - execution, (2)7-7
 - format, (2)7-6, 7-7
- REWIND
 - description, (2)5-17
 - format, (2)5-16
- SAVE
 - description, (2)2-16
 - format, (2)2-16
- source, listings, (3)1-8.1
- statement function definition, format, (2)7-3
- STOP, format, (2)4-15
- SUBROUTINE, format, (2)7-6
- terminal
 - description, (2)4-10
 - execution, (2)4-13
- two-branch arithmetic IF
 - description, E-6
 - format, E-6
- type
 - CHARACTER, (2)2-8
 - COMPLEX, (2)2-7
 - data length, E-11
 - description, (2)2-7
 - double declaration, E-10
 - DOUBLE PRECISION, (2)2-7
 - format, (2)2-7
 - INTEGER, (2)2-7
 - LOGICAL, (2)2-7
 - REAL, (2)2-7
 - with data length, format, E-11
- unconditional GO TO
 - execution, (2)4-2
 - format, (2)4-1
- WRITE, format, (2)5-8

Statement function, (1)4-3

- definition statement, format, (2)7-3
- dummy argument, (2)7-3
- execution, (1)4-4
- referencing, (1)4-4
- restrictions, (1)4-4
- symbolic name, (1)2-28

Statement numbers, table, (3)1-9

Static Space Table, (3)1-11

Status, vector, of subprograms, B-1

STOP statement, format, (2)4-15

Storage sequence

- array, (1)2-12.1, 2-13
- associated, (1)2-19
- common block, (2)2-14
- description, (1)2-19
- size, (1)2-19

Subprogram

- block data
 - description, (1)4-1, (2)7-11
 - symbolic name, (1)2-26
- description, (1)1-12, 4-1, (2)7-1
- entry, procedure
 - referencing, (2)7-9
- function
 - data type, (1)2-2
 - description, (1)4-3, (2)7-2
 - entry association, (2)7-9
 - restrictions, (1)4-7
- non-FORTRAN, (1)4-9
- procedure, (1)4-2
- specification, (1)4-1
- subroutine
 - description, (1)4-2
 - restrictions, (1)4-3
- types, (1)4-1
- vector status of, B-1

Subroutine

- actual arguments, (1)4-2
- CLEARFI, (3)2-15
- reference, (2)7-5
- SENSEFI, (3)2-15
- SETFI, (3)2-15
- subprogram
 - description, (1)4-2
 - restrictions, (1)4-3
- symbolic name, (1)2-26
- user control, (2)5-26, 5-28

SUBROUTINE statement, format, (2)7-6

Subscript

- evaluation, (1)2-16
- values, (1)2-15

Substring

- character
 - actual argument, (1)4-11
 - description, (1)2-18
- designator
 - description, (1)2-18
 - format, (1)2-18

Summary, rules of interpretation, (1)3-20

Symbolic debug package, G-1

Symbolic names

- array, (1)2-26
- block data subprogram, (1)2-26
- classes, (1)2-25
- common block, (1)2-25

Symbolic names (continued)

- constant, (1)2-27
- description, (1)1-6, 2-23
- dummy procedure, (1)2-28
- external function, (1)2-25
- intrinsic function, (1)2-28
- main program, (1)2-26
- scope, (1)2-24
- statement function, (1)2-28
- subroutine, (1)2-26
- variable, (1)2-27

SYMDEBUG library routine, G-1

Syntactic items, (1)1-5, 1-7

Syntax, FORTRAN statement forms, (1)1-2

T, TL, and TR

- edit descriptors, (2)6-8
- editing, (2)6-9

Table

- block names and lengths in octal, (3)1-11
- external names, (3)1-12
- loops encountered, (3)1-12
- names encountered, (3)1-9
- program unit page, (3)1-12.1
- Stack Space, (3)1-11
- statement numbers, (3)1-9
- Static Space (3)1-11

Temporary

- scalar, (3)2-3
- variable reference, local, F-1

Term

- description, (1)3-5
- logical
 - form, (1)3-15
 - value, (1)3-16

Terminal

- line, (1)1-9, 1-10
- parameter, (2)4-11
- statement
 - description, (2)4-10
 - execution, (2)4-13

Termination, relieve, I-2

Time and date functions, B-5

Totally associated entities, (1)2-19

Trace, flow

- directives, (3)1-21
- enable/disable, (3)1-22
- routine options, (3)1-23

Traceback, error, F-1

Transfer

- data
 - dataset position after, (2)5-15
 - dataset position before, (2)5-5
 - description, (2)5-12, 5-13
 - direction of, (2)5-12
 - execution, (2)5-12
 - formatted, (2)5-14
 - unformatted, (2)5-14
- into range of a DO-loop, (2)4-14

Transmission, argument, F-1

Trigonometric functions, B-3

Two-branch arithmetic IF statement

- description, E-6
- format, E-6

Types

- array declarator, (1)2-11
- character, dummy argument, (1)4-11
- conversion
 - assignment statements, (1)3-8
 - Boolean, (1)3-10
 - complex, (1)3-10
 - description, (1)3-9
 - double-precision, (1)3-9
 - functions, B-4
 - integer, (1)3-9
 - real, (1)3-9
- data, (1)2-1
 - array element, (1)2-2
 - edit descriptors with, (2)6-5
 - function, (1)2-2
 - function subprogram, (1)2-2
 - of arithmetic expression, (1)3-6
- field, (3)1-10
- statement
 - CHARACTER, (2)2-8
 - COMPLEX, (2)2-7
 - data length, E-11
 - description, (2)2-7
 - double declaration, format, E-10
 - DOUBLE PRECISION, (2)2-7
 - format, (2)2-7
 - INTEGER, (2)2-7
 - LOGICAL, (2)2-7
 - REAL, (2)2-7
 - with data length, format, E-11

Unblocked datasets, H-1

Unconditional GO TO statement

- execution, (2)4-2
- format, (2)4-1

Undefined

- dummy argument, (1)4-11
- entities, (1)2-22
- variable or array element, (1)2-21

Unformatted

- data transfer, (2)5-14
- records, (2)5-2

Unit

- description, (2)5-6
- identifier
 - external, (2)5-6
 - format, (2)5-6
- identifying, (2)5-13
- program
 - description, (1)1-10, (2)7-1
 - page table, (3)1-12.1
 - specification, program (2)7-1
 - specifier, external, (2)5-18

UNIT

- function (CFT extension), (2)5-33
- utility function, (2)5-33

UNSAFEIF directive, (3)1-24.1

Unsigned constant, (1)2-3

Upper bounds of dimension declarators, (1)2-11

Uppercase/lowercase conversion, (1)1-5

Use of

- array names, (1)2-17
- optimized routines by CFT, (3)2-12

User
 control subroutine argument lists,
 (2)5-27
 control subroutines, (2)5-26, 5-28
 Using optimized routines, (3)2-11
 Utility
 function
 LENGTH, (2)5-34
 UNIT, (2)5-33
 procedures, (1)4-6, C-1

Values
 function, F-1
 logical
 disjunct, (1)3-16
 expression, (1)3-16
 factor, (1)3-16
 term, (1)3-16
 subscript, (1)2-15
 Value separators, (2)6-24
 Variable
 defined, (1)2-21
 description, (1)2-9
 DO, (2)4-10
 dummy arguments, (1)4-11
 initially defined, (1)2-21
 name, (1)2-27
 NAMELIST input, (2)5-25
 reference, local temporary, F-1
 symbolic name, (1)2-27
 undefined, (1)2-21
 used in a reduction array operation,
 (3)2-3

Vector
 array reference, (3)2-3
 pseudo, (3)2-3
 status of subprograms, B-1
 VECTOR directive
 description, (3)1-16
 format, (3)1-16
 Vectorizable DO-loops, (3)2-1
 Vectorization
 aids, B-6
 control directives, (3)1-16
 efficiency, (3)2-11
 guidelines, (3)2-16
 qualifications, (3)2-1
 with arrays, (3)2-10
 VFUNCTION directive
 description, (3)1-19
 format, (3)1-19

Width, field, (2)6-7
 WRITE statement, format, (2)5-8
 Writing, (2)5-1
 WRITMS routine, E-14

X
 edit descriptor, (2)6-8
 editing, (2)6-9

Z (hexadecimal)
 edit descriptor, (2)6-20
 editing (CFT extension), (2)6-20
 Z option, CFT, G-1